Classification

Logistic Regression

- Problem
  - **Data**: observed pairs (x, y), where $x \in \mathcal{X}$ (**input**) & $y \in \mathcal{Y}$ (**output**)
    - Binary classification: $\mathcal{Y} = \{-1, +1\} \vee \{0,1\}$
    - Multiclass classification: $\mathcal{Y} = \{1, \dots, K\}$
  - **Goal**: find a classifier f that can map input x to class y:

$$f: \mathcal{X} \to \mathcal{Y} \quad \text{s.t.} \quad \forall (x, y): y = f(x)$$

- Model
  - Assumption: **Independent & Identically distributed [i.i.d.]**

$$(x_i, y_i) \sim \mathcal{P} \quad | \quad i = 1, \dots, m$$

    - = "The future should look like the past."

  - Model

$$\hat{y} = g(w^T x)$$

    - $g(w^T x)$: an activation function that converts $w^T x$ to binary values (See more in Deep Learning)
    - Sigmoid Function

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

    - Derivative

$$g'(z) = g(z)\big(1 - g(z)\big)$$

- Algorithm
  - Cost Function

$$\mathcal{L}(\hat{y}, y) = -(y \ln \hat{y} + (1 - y) \ln(1 - \hat{y}))$$

    - If $y = 1 \longrightarrow \mathcal{L}(\hat{y}, y) = -\ln \hat{y} \longrightarrow$ want "$\mathcal{L} \downarrow \ \leftrightarrow \ \hat{y} \uparrow$" $\longrightarrow \hat{y} = 1$
    - If $y = 0 \longrightarrow \mathcal{L}(\hat{y}, y) = -\ln(1 - \hat{y}) \longrightarrow$ want "$\mathcal{L} \downarrow \ \leftrightarrow \ \hat{y} \downarrow$" $\longrightarrow \hat{y} = 0$

  - Gradient Descent

$$w_j \coloneqq w_j - \alpha(y - \hat{y})x_j$$

- Probabilistic Interpretation
  - Assumptions

$$p(y|x, w) = \hat{y}^y (1 - \hat{y})^{1-y}$$

    - $P(y = 1|x, w) = \hat{y}$
    - $P(y = 0|x, w) = 1 - \hat{y}$

  - Likelihood Function

$$L(w) = \prod_{i=1}^{m} \left(\hat{y}^{(i)}\right)^{y^{(i)}} \left(1 - \hat{y}^{(i)}\right)^{1-y^{(i)}}$$

  - Log Likelihood

$$\begin{aligned} l(w) &= \sum_{i=1}^{m} \left(y^{(i)} \ln \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \ln\left(1 - \hat{y}^{(i)}\right)\right) \\ &= -\sum_{i=1}^{m} \mathcal{L}(\hat{y}, y) \end{aligned}$$

  - MLE

$$\frac{\partial l(w)}{\partial w_j} = \left(\frac{y}{g(w^T x)} - \frac{1 - y}{1 - g(w^T x)}\right) \frac{\partial g(w^T x)}{\partial w_j}$$

$$= \left(\frac{y}{g(w^T x)} - \frac{1-y}{1-g(w^T x)}\right) g(w^T x)\left(1 - g(w^T x)\right) \frac{\partial (w^T x)}{\partial w_j}$$
$$= (y - \hat{y}) x_j$$

- Gradient Descent

$$w_j \coloneqq w_j - \alpha (y - \hat{y}) x_j$$

K-Nearest Neighbors

- <u>Problem</u>
  - Classification
  - **Memory-based**: stores all available cases & classifies new cases based on a **similarity measure**

- <u>Algorithm</u>:

  For a new input x,

  1. Return the k points **closest** to x, indexed as $x_{i_1}, \dots, x_{i_k}$.

  2. Return the majority votes of $y_{i_1}, \dots, y_{i_k}$.

  - k
    - ♦ Smaller k → smaller training err but could lead to overfitting
    - ♦ Larger k → more stable predictions due to voting

- **Distances** (how to measure "closest")

  - **Euclidean distance**: default measurement

$$\|u - v\|_2 = \left(\sum_{i=1}^{n}(u_i - v_i)^2\right)^{\frac{1}{2}}$$

  - **Manhattan distance**:

$$\|u - v\|_1 = \sum_{i=1}^{k}|x_i - y_i|$$

  - **Minkowski l_p**: variation on Euclidean

$$\|u - v\|_p = \left(\sum_{i=1}^{n}|u_i - v_i|^p\right)^{\frac{1}{p}} \quad | \quad p \in [1, \infty)$$

  - **Hamming distance**: for categorical variables

$$u = v \Rightarrow D = 0$$
$$u \neq v \Rightarrow D = 1$$

  - **Edit distance**: for strings

    #modifications required to transform one string to the other

  - **Correlation distance**: for signals

    How correlated 2 vectors are for signal detection

Gaussian Discriminant Analysis

- Problem
  - **Discriminative Learning Algorithms**

$$\text{model } p(y|x) \text{ directly} \quad (X \to Y)$$

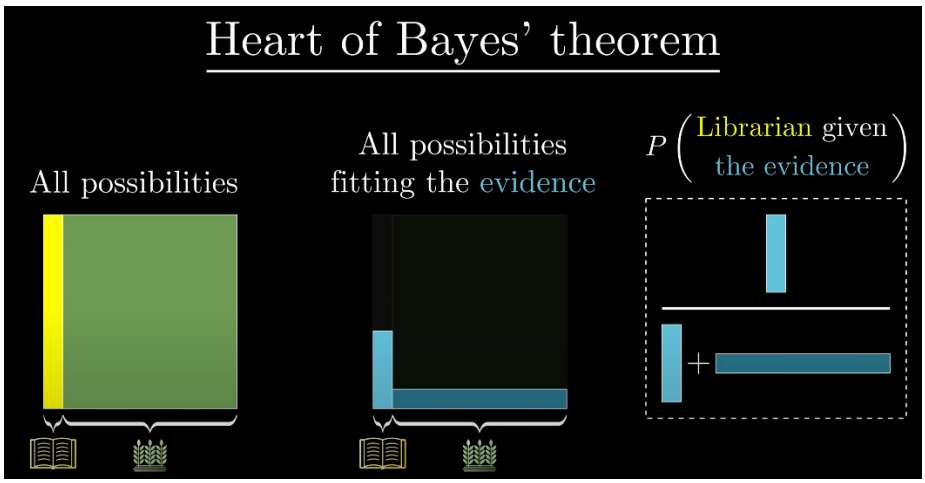
  - **Generative Learning Algorithms**

$$\text{model } p(x|y) \ \& \ p(y) \quad \left(\text{use Bayes Theorem to get } p(y|x)\right)$$

- **Bayes Theorem**: Core of GLAs

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

  - **Prior**: $p(y)$
  - **Posterior**: $p(y|x)$
  - **The Brilliant Intuition for Bayes Theorem**



Cited from 3Blue1Brown

1. We want to determine whether Steve is a librarian $[\boldsymbol{y}]$ instead of a farmer, given that he is a shy boy $[\boldsymbol{x}]$. ⟹ We need to know the possibility of this $[\boldsymbol{p(y|x)}]$.

2. We notice that there are many more farmers $[\boldsymbol{p(y)}]$ than librarians $[\boldsymbol{p(\neg y)}]$ in the sample pool **[Dataset]**.

3. We also notice that the ratio of shy people among librarians $[\boldsymbol{p(x|y)}]$ is much higher than that among farmers $[\boldsymbol{p(x|\neg y)}]$.

4. We add all the shy people up $[\boldsymbol{p(x) = p(y)p(x|y) + p(\neg y)p(x|\neg y)}]$, whether they are librarians or farmers.

5. We calculate how many of these shy people are actually librarians $[\boldsymbol{p(y)p(x|y)}]$ and calculate the proportion of these librarians among the shy people, which will be our probability $[\boldsymbol{p(y|x)}]$.

- Simplification of Bayes Theorem in ML:

  ∵ we are trying to find the output y with the highest probability given x

  ∴ we can simplify Bayes Theorem for this purpose:

$$\underset{y}{\operatorname{argmax}}\, p(y|x) = \underset{y}{\operatorname{argmax}}\, p(x|y)p(y)$$

Model

- Assumption: **Multivariate Gaussian Distribution**

$$p(x|\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} e^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$$

  It is literally the same as normal Gaussian but with vector parameters:

- ♦ Mean vector: $\mu \in \mathbb{R}^n$
- ♦ Covariance matrix: $\Sigma \in \mathbb{R}^{n \times n}$

As a comparison, here is the univariate version:

$$p(x|\mu, \Sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Model

$$y \sim \text{Bernoulli}(\phi)$$
$$x|y = 0 \sim N(\mu_0, \Sigma)$$
$$x|y = 1 \sim N(\mu_1, \Sigma)$$

Algorithm & Probabilistic Interpretation

$$p(y) = \phi^y (1-\phi)^{1-y}$$
$$p(x|y=0) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} e^{\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)}$$
$$p(x|y=1) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} e^{\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)}$$

- Log likelihood

$$l(\phi, \mu_0, \mu_1, \Sigma) = \ln \prod_{i=1}^{m} p\left(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma\right) p\left(y^{(i)}; \phi\right)$$

- MLE

$$\phi = \frac{1}{m}\sum_{i=1}^{m} \mathrm{I}\{y^{(i)} = 1\}$$
$$\mu_0 = \frac{\sum_{i=1}^{m} \mathrm{I}\{y^{(i)} = 0\} x^{(i)}}{\sum_{i=1}^{m} \mathrm{I}\{y^{(i)} = 0\}}$$
$$\mu_1 = \frac{\sum_{i=1}^{m} \mathrm{I}\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^{m} \mathrm{I}\{y^{(i)} = 1\}}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \mu_{y^{(i)}}\right)\left(x^{(i)} - \mu_{y^{(i)}}\right)^T$$

- **GDA vs LogReg**
  - GDA
    - Makes **stronger** modeling assumptions about data
    - Data efficient when assumptions (Gaussian distributions) are approximately correct
  - LogReg
    - Makes **weaker** modeling assumptions about data
    - Data efficient when assumptions (Gaussian distributions) are not necessarily correct (e.g. $x|y \sim \text{Poisson}(\lambda_1)$ instead of $N(\mu_0, \Sigma)$)

Naïve Bayes Classifier

- Problem
  - GDA vs NB
    - GDA: x = continuous, real-valued vectors
    - NB: x = discrete-valued vectors (e.g. text classification)

  - **Text Encoding** (more in RNN)

    We encode a text sentence into a vector of the same length as our **dictionary** (like a Python dictionary with vocab & their indices as key-value pairs). The encoded sentence looks like:

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 1 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \text{a} \\ \text{abandon} \\ \vdots \\ \text{pewdiepie} \\ \vdots \\ \text{subscribe} \\ \text{to} \\ \vdots \\ \text{zuck} \end{matrix}$$

    The original sentence was "Subscribe to Pewdiepie!", and this text encoding method uses lowercases, throws punctuations and ignores the order of the sentence. This is convenient in some cases (e.g. spam email classification) but awful in many other cases (e.g. news/report-writer bots).

    Notice that $x \in \{0,1\}^{\text{len(dict)}}$. Why notice this? Because we now have $2^{\text{len(dict)}}$ possible outcomes for x. When we have a dictionary of over 20000 words, we will have a $(2^{20000} - 1)$-dimensional encoded vector. Not good for our PCs.

- Model
  - Assumption: **Conditional Independence**

$$\forall j \neq i: \ p(x_i|y, x_j) = p(x_i|y)$$

    - Intuition: Given y as the condition, x_i is independent of x_j.
    - Example: In the case of spam email classification, if we know that the email is spam, then whether or not "pewdiepie" is in the sentence does not change our belief of whether or not "subscribe" is in the sentence.
    - Further extension: we can simplify our $p(x|y)$ into:

$$p(x_1, \dots, x_{\text{len(dict)}}|y) = \prod_{i=1}^{n} p(x_i|y)$$

  - Model

$$\phi_{i|y=1} = p(x_i = 1|y = 1)$$
$$\phi_{i|y=0} = p(x_i = 1|y = 0)$$
$$\phi_y = p(y = 1)$$

- Algorithm & Probabilistic Interpretation
  - Joint likelihood

$$\mathcal{L}(\phi_y, \phi_{i|y=1}, \phi_{i|y=0}) = \prod_{i=1}^{m} p(x^{(i)}, y^{(i)})$$

    - $\phi_{i|y=0}$: the probability of the existence of word i in Non-spam emails.
  - MLE

$$\phi_{j|y=1} = \frac{\sum_{i=1}^{m} \mathrm{I}\left\{x_j^{(i)} = 1 \ \wedge \ y^{(i)} = 1\right\}}{\sum_{i=1}^{m} \mathrm{I}\{y^{(i)} = 1\}}$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^{m} \mathrm{I}\left\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\right\}}{\sum_{i=1}^{m} \mathrm{I}\{y^{(i)} = 0\}}$$

$$\phi_y = \frac{\sum_{i=1}^{m} \mathrm{I}\left\{y^{(i)} = 1\right\}}{m}$$

- $\phi_{j|y=0}$: the fraction of Non-spam emails with the word j in it.

- Prediction

$$p(y = 1|x_{new}) = \frac{p(x_{new}|y = 1)p(y = 1)}{p(x_{new})}$$

$$= \frac{\prod_{i_{new}=1}^{n_{new}} p\left(x_{i_{new}}|y = 1\right)p(y = 1)}{\prod_{i_{new}=1}^{n_{new}} p\left(x_{i_{new}}|y = 1\right)p(y = 1) + \prod_{i_{new}=1}^{n_{new}} p\left(x_{i_{new}}|y = 0\right)p(y = 0)}$$

- The equation is tedious but very intuitive. The y with the higher posterior will be chosen as the final prediction.

- Apply NB in GDA cases?

  Discretize: Cut the continuous, real-valued x into small intervals and label them with a discrete-valued scale.

• Laplace Smoothing

- Problem

  What if there is a new word "mrbeast" in the email for prediction that our NB classifier has never learnt before?

  A human would look it up on a dictionary, and so would NB.

  Assume the word "mrbeast" is the 1234th word in the dictionary, then:

$$\phi_{1234|y=1} = \frac{\sum_{i=1}^{m} \mathrm{I}\left\{x_{1234}^{(i)} = 1 \wedge y^{(i)} = 1\right\}}{\sum_{i=1}^{m} \mathrm{I}\{y^{(i)} = 1\}} = 0$$

$$\phi_{1234|y=0} = \frac{\sum_{i=1}^{m} \mathrm{I}\left\{x_{1234}^{(i)} = 1 \wedge y^{(i)} = 0\right\}}{\sum_{i=1}^{m} \mathrm{I}\{y^{(i)} = 0\}} = 0$$

Yes. NB believes that the probability of seeing this word in either Spam or Non-spam emails is 0, and therefore it would predict that:

$$p(y = 1|x_{new}) = \frac{0}{0}$$

During prediction, if NB never encountered a word j, there will always be a $\phi_j = 0$ that ruins the entire prediction. How do we estimate the unknown?

- Algorithm

$$\phi_j = \frac{\sum_{i=1}^{m} \mathrm{I}\{z^{(i)} = j\} + 1}{m + k}$$

where $k =$ #features.

This will still satisfy the basic sum rule of probability:

$$\sum_{j=1}^{k} \phi_j = \frac{m + k}{m + k} = 1$$

The estimates in NB will now become:

$$\phi_{j|y=1} = \frac{\sum_{i=1}^{m} \mathrm{I}\left\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\right\} + 1}{\sum_{i=1}^{m} \mathrm{I}\{y^{(i)} = 1\} + 2}$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^{m} \mathrm{I}\left\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\right\} + 1}{\sum_{i=1}^{m} \mathrm{I}\{y^{(i)} = 0\} + 2}$$

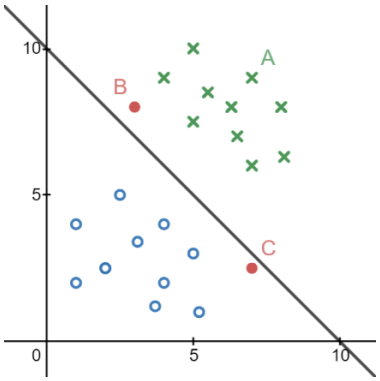
- Example

In the case of "mrbeast", it will become $\frac{1}{2}$ for both $\phi_{1234|y=0}$ and $\phi_{1234|y=1}$.

Now NB can give a very fair estimate with new words.

Support Vector Machine

- Problem

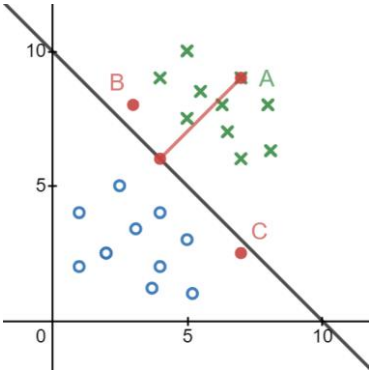


This is a binary classification.

The circles & crosses are training examples with 2 different labels.

The black line is the decision boundary from the classifier that can classify "circle" and "cross".

For points like A that are distant from the classifier, we are quite confident that they belong to "cross".



HOWEVER, what about B and C that are super close to the decision boundary? Based on this classifier, B belongs to "cross" and C belongs to "circle", but how confident are we about this classifier? What if this classifier is just slightly off and C is actually a "cross"?

- SVM vs Other Algorithms
  - Other algorithms

    ⇒ Learn: **Differences**

    ⇒ Goal: find a **classifier** that can **map input to class**

  - SVM

    ⇒ Learn: **Similarities**

    ⇒ Goal: find the **optimal hyperplane** that separates the classes

• Margins

- **Functional Margin**

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x + b) \quad | \quad y \in \{-1,1\}$$

- Intuition: $\hat{\gamma}^{(i)} \uparrow\uparrow \rightarrow$ confidence $\uparrow\uparrow$

$$\text{When } y = 1 \rightarrow w^T x + b \gg 0$$

$$\text{When } y = -1 \rightarrow w^T x + b \ll 0$$

- Problem:

  If $w \rightarrow kw$ & $b \rightarrow kb$ $(k > 0)$, then $g(w^T x + b) \rightarrow g\big(k(w^T x + b)\big)$.

  However, recall that $g(z)$ follows:

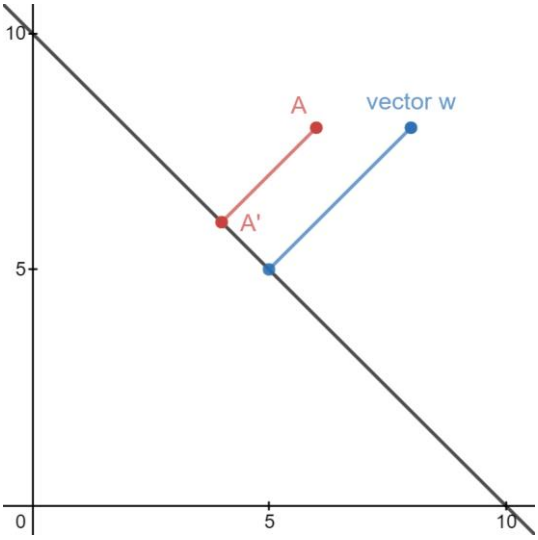
$$g(z) = \begin{cases} -1 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \end{cases}$$

  z or kz makes no difference for $g(z)$ at all.

  HOWEVER, functional margin DOES change by a factor of k here, meaning that a large functional margin does NOT necessarily represent a confident prediction.

- **Geometric Margin**

  Refer back to the figure above. If we want to find the distance between point A and the decision boundary (i.e. $AA' = \gamma^{(i)}$), what should we do?



We normalize w to find the unit vector $\frac{\boldsymbol{w}}{\|w\|}$, and we also have $A = \boldsymbol{x^{(i)}}$.

$$\because AA' \parallel \boldsymbol{w}$$

$$\therefore A' = \boldsymbol{x^{(i)}} - \gamma^{(i)} \frac{\boldsymbol{w}}{\|w\|}$$

$$\because A' \text{ is on } \boldsymbol{w^T x} + b = 0$$

$$\therefore \boldsymbol{w^T} A' + b = 0$$

$$\Rightarrow \boldsymbol{w^T x^{(i)}} + b = \boldsymbol{w^T} \gamma^{(i)} \frac{\boldsymbol{w}}{\|w\|}$$

$$\Rightarrow \gamma^{(i)} = \left(\frac{\boldsymbol{w}}{\|w\|}\right)^T \boldsymbol{x^{(i)}} + \frac{b}{\|w\|}$$

If we generalize it with both classes of $y^{(i)}$ (and ignore vector notation):

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{w}{\|w\|}\right)^T x^{(i)} + \frac{b}{\|w\|}\right)$$

- # Lagrange Duality

  - **Constrained optimization problem**

$$\min_{w} f(w) \quad \text{s.t.} \quad h_i(w) = 0 \; \forall i \in \{1, \dots, m\}$$

    - ♦ Intuition: minimize $f(w)$ on the set $\{w | h_i(w) = 0 \; \forall i \in \{1, \dots, m\}\}$.

  - **Lagrangian**

$$\mathcal{L}(w, \beta) = f(w) + \sum_{i=1}^{m} \beta_i h_i(w)$$

    where $\beta_i =$ Lagrange multipliers. We solve this problem by:

$$\frac{\partial \mathcal{L}}{\partial w_i} = 0$$
$$\frac{\partial \mathcal{L}}{\partial \beta_i} = 0$$

  - **Generalized COP**

$$\min_{w} f(w) \quad \text{s.t.} \quad h_i(w) = 0 \; \forall i \in \{1, \dots, m\}$$
$$g_i(w) \leq 0 \; \forall i \in \{1, \dots, n\}$$

    - ♦ Intuition: add an inequality constraint to the original OP

  - **Generalized Lagrangian**

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{m} \beta_i h_i(w) + \sum_{i=1}^{n} \alpha_i g_i(w)$$

  - **Primal OP**

$$p^* = \min_{w} \theta_{\mathcal{P}}(w) = \min_{w} \max_{\alpha, \beta: \, \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta)$$

    - ♦ Intuition: Under the 2 primal constraints above, the maximum of Generalized Lagrangian ($\theta_{\mathcal{P}}(w)$) is basically just $f(w)$ as long as $\alpha_i \geq 0$:

$$\sum_{i=1}^{m} \beta_i h_i(w) \to \sum_{i=1}^{m} \beta_i \cdot 0 \to 0$$

$$\sum_{i=1}^{n} \alpha_i g_i(w) \xrightarrow{\alpha_i \geq 0,\ \ g_i(w) \leq 0} \sum_{i=1}^{n} (+0 \cdot -0) \to 0$$

- **Dual OP**

$$d^* = \max_{\alpha,\beta:\, \alpha_i \geq 0} \theta_{\mathcal{D}}(\alpha, \beta) = \max_{\alpha,\beta:\, \alpha_i \geq 0} \min_{w} \mathcal{L}(w, \alpha, \beta)$$

  - Intuition: Even though this seems very similar to Primal OP, their values are not necessarily equal. Instead, they are:

$$d^* \leq p^*$$

    The proof is very simple. For any function $f(x, y)$:

$$\min_{w} f(x, w) \leq f(x, y) \leq \max_{v} f(v, y)$$

$$\min_{u} f(u, y) \leq f(x, y) \leq \max_{t} f(x, t)$$

    Therefore, the following also holds:

$$\max_{x} \left(\min_{w} f(x, w) \right) \leq \min_{y} \left(\max_{v} f(v, y) \right)$$

    which is basically saying that "$\max \min \leq \min \max$" for all multivariate functions, including our Lagrangian.

- **Karush-Kuhn-Tucker Conditions [KKT]**

  - Under the above assumptions, there MUST exist w^*, α^*, β^* so that

    $\Rightarrow$ w^* is the solution to the Primal OP

    $\Rightarrow$ α^*, β^* are the solution to the Dual OP

    $\Rightarrow$ $p^* = d^* = \mathcal{L}(w^*, \alpha^*, \beta^*)$

- ♦ KKT Conditions: w^*, α^*, β^* must satisfy:

$$\frac{\partial}{\partial w_i}\mathcal{L}(w^*, \alpha^*, \beta^*) = 0 \quad \forall i \in \{1, \dots, m\}$$

$$\frac{\partial}{\partial \beta_i}\mathcal{L}(w^*, \alpha^*, \beta^*) = 0 \quad \forall i \in \{1, \dots, m\}$$

$$\alpha_i^* g_i(w^*) = 0 \quad \forall i \in \{1, \dots, n\}$$

$$g_i(w^*) \leq 0 \quad \forall i \in \{1, \dots, n\}$$

$$\alpha_i^* \geq 0 \quad \forall i \in \{1, \dots, n\}$$

• Optimal Margin Classifiers

- ○ Primal OP for SVM:

$$\min_{\gamma, w, b} \frac{1}{2}\|w\|^2 \quad \text{s.t.} \quad y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 \; \forall i \in \{1, \dots, m\}$$

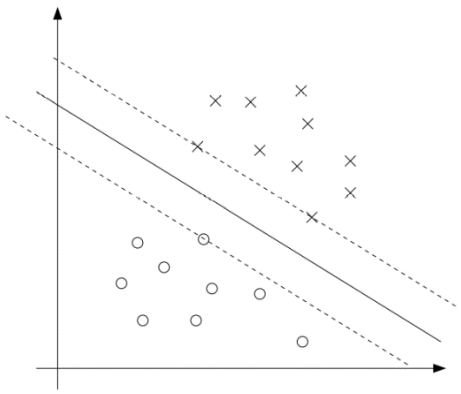
$\frac{1}{2}$ simplifies further differentiations.

An alternative way to write the constraints would be:

$$g_i(w) = 1 - y^{(i)}\left(w^T x^{(i)} + b\right) \leq 0$$

Note that $\alpha_i > 0$ iff $\hat{\gamma}^{(i)} = 1$ (i.e. $g_i(w) = 0$).

- ○ **Support Vectors**:



The 2 dotted lines are the only lines where we can get $\hat{y}^{(i)} = 1 \rightarrow \alpha_i > 0$.

The 3 points on the 2 dotted lines are **Support Vectors**.

This always holds: #SVs $\ll$ #xs.

- ♦ Intuition
  - ⇒ Suppose we have 2 classes: "apple" (◯) and "lemon" (×)
  - ⇒ The ◯ on the dotted line = yellow apple
  - ⇒ The × on the dotted line = red lemon
  - ⇒ **Support Vectors**: the data of class A that has the most similar features to class B.
  - ⇒ If we find SVs, we can accurately separate classes by choosing the decision boundary in the exact middle between the dotted lines.

- ○ Lagrangian for SVM:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)} \left(w^T x^{(i)} + b \right) - 1 \right]$$

$$\nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0$$

$$\Rightarrow w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

$$\nabla_b \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

$$\Rightarrow \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

$$\Longrightarrow \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \left(x^{(i)} \right)^T x^{(j)}$$

- Dual OP for SVM:

$$\max_{\alpha} \mathcal{L}(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

$$\text{s.t. } \alpha_i \geq 0 \quad \& \quad \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

After solving α^*s, we can solve w^* as:

$$w^* = \sum_{i=1}^{m} \alpha_i^* y^{(i)} x^{(i)}$$

And we can then solve b^* as:

$$b^* = -\frac{\max\limits_{i:\, y^{(i)}=-1} w^{*T} x^{(i)} + \min\limits_{i:\, y^{(i)}=1} w^{*T} x^{(i)}}{2}$$

- Simplified Prediction:

Recall that in order to make a prediction, we need to calculate $w^T x_{new} + b$.

If this is significantly bigger than 0, then we predict $y = 1$.

This can be much simplified as:

$$w^T x_{new} + b = \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \right)^T x_{new} + b$$

$$= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x_{new} \rangle + b$$

Since $\alpha_i \neq 0$ ONLY for support vectors, the only inner products we need to calculate are $\langle x_{SV}, x_{new} \rangle$.

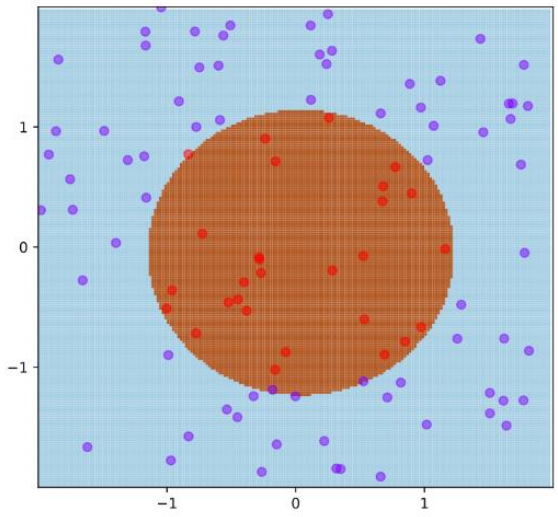
- Algorithm

  1. Find the 2 hyperplanes that separate the data & contain no data in between

  2. Maximize the margin until $\hat{y}^{(i)} = 1$ (i.e. until the hyperplanes hit the SVs)

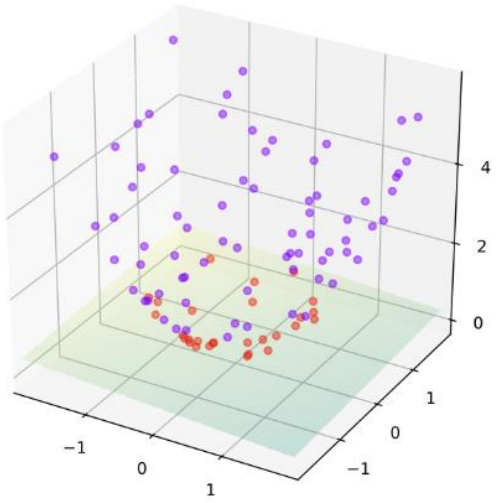
  3. Choose the plane in the exact middle as the decision boundary

- **Kernels**
  - Problem: **Nonlinear datasets**



    It takes tremendous amount of efforts to make **linear** learning algorithms to learn a **nonlinear** decision boundary.

  - Solution:
    - Step 1: **Feature mapping**: $\phi: \mathcal{X} \to \mathcal{V}$
    - Step 2: **Kernel method**: $K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$
    - Step 3: **Replacement**: replace all $\langle \mathbf{x}, \mathbf{x}' \rangle$ in the algorithm with $K(\mathbf{x}, \mathbf{x}')$



The feature mapping can transform raw input into new space (or new dimension in this case) so that our SVM can proceed to find the optimal decision boundary, which is a 2D plane separating purple and red.

- Example **(feature mapping → kernel)**:
  - Based on the graphs above, we have $\mathbf{x} = (x_1, x_2)$. Since the decision boundary in 2D seems like a circle, we define such a feature mapping:

$$\phi\big((a,b)\big) = \begin{bmatrix} a \\ b \\ a^2 + b^2 \end{bmatrix}$$

  - Step 1: Feature mapping:

$$\phi(\mathbf{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{bmatrix}, \qquad \phi(\mathbf{x}') = \begin{bmatrix} x_1' \\ x_2' \\ {x_1'}^2 + {x_2'}^2 \end{bmatrix}$$

  - Step 2: Kernel method:

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= \phi(\mathbf{x})^T \phi(\mathbf{x}') \\ &= x_1 x_1' + x_2 x_2' + (x_1^2 + x_2^2)\big({x_1'}^2 + {x_2'}^2\big) \\ &= \mathbf{x} \cdot \mathbf{x}' + \|\mathbf{x}\|^2 \|\mathbf{x}'\|^2 \end{aligned}$$

  - Step 3: Replacement: replace all $\langle \mathbf{x}, \mathbf{x}' \rangle$ in the algorithm with $K(\mathbf{x}, \mathbf{x}')$

- Example **(kernel → feature mapping)**:
  - It is very costly to do computations with high-dimension vectors like $\phi(\mathbf{x})$. However, it is much easier to do computations with $K(\mathbf{x}, \mathbf{x}')$. Thus, we can get SVMs to learn in high-dimension feature space without ever learning the actual $\phi(\mathbf{x})$.
  - Suppose the kernel is:

$$\begin{aligned} K(x,y) &= (x^T y)^2 \\ &= \left(\sum_{i=1}^{n} x_i y_i\right)\left(\sum_{j=1}^{n} x_j y_j\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j y_i y_j \\ &= \sum_{i,j=1}^{n} (x_i x_j)(y_i y_j) \\ &= \phi(x)^T \phi(y) \end{aligned}$$

  - Then we get the feature mapping:

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ \vdots \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}$$

- Intuition:

  Kernel = similarity between feature mappings of 2 data

  - If $\phi(x)$ & $\phi(y)$ are close, then $K(x,y) = \phi(x)^T \phi(y)$ is large
  - If $\phi(x)$ & $\phi(y)$ are far apart, then $K(x,y) = \phi(x)^T \phi(y)$ is small

- Types of Kernels:

  - **Gaussian Kernel** (when there is no prior knowledge of data)

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

  - **Gaussian Radial Basis Function [RBF] Kernel**

$$K(x,y) = e^{-\gamma\|x-y\|^2}, \gamma > 0$$

  - **Laplace RBF Kernel**

$$K(x,y) = e^{-\frac{\|x-y\|}{\sigma}}$$

  - **Polynomial Kernel** (image processing)

$$K(x,y) = (x \cdot y + 1)^d$$

  - **Hyperbolic Tangent Kernel** (neural networks)

$$K(x,y) = \tanh(\kappa x \cdot y + c)$$

  - **Sigmoid Kernel**

$$K(x,y) = \sigma(wx^T y + b)$$

  - **Bessel Function of 1st Kind Kernel** (remove cross term in math)

$$K(x,y) = \frac{J_{v+1}(\sigma\|x-y\|)}{\|x-y\|^{-n(v+1)}}$$

    where

$$J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\,\Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha}$$

  - **ANOVA RB Kernel** (regression)

$$K(x,y) = \sum_{k=1}^{n} e^{\left(-\sigma\left(x^k - y^k\right)^2\right)^d}$$

- **Linear Splines Kernel** (text categorization, large sparse data)

$$K(x,y) = 1 + xy + xy\min(x,y) - \frac{x+y}{2}\min(x,y)^2 + \frac{1}{3}\min(x,y)^3$$

- **Mercer Theorem:** Suppose $K: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$. In order for K to be a valid (Mercer) kernel, it is necessary & sufficient that for any $\{x^{(1)}, \dots, x^{(m)}\}$, the corresponding **kernel matrix** is **symmetric, positive semi-definite**.
  - **Kernel matrix**: a $m \times m$ matrix K with the entries given as $K_{ij} = K(x^{(i)}, x^{(j)})$ for some finite set of m points $\{x^{(1)}, \dots, x^{(m)}\}$
  - **Symmetric:**

$$\begin{aligned} K_{ij} &= K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)}) \\ &= \phi(x^{(j)})^T \phi(x^{(i)}) = K(x^{(j)}, x^{(i)}) = K_{ji} \end{aligned}$$

  - **Positive semi-definite**: let $\phi_k(x)$ be the kth coordinate of $\phi(x)$. For any vector z:

$$\begin{aligned} z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\ &= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j \\ &= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)})^T \phi_k(x^{(j)})\, z_j \\ &= \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \\ &\geq 0 \end{aligned}$$

- **Non-Separable Data → Regularization**
  - Problems:
    - 1. Feature mapping to a higher dimension does not guarantee that the data is separable for classification.
    - 2. Looking for a decision boundary may not be the exact goal of classification ← outliers cause significant changes to the hyperplane.
  - **OP**:
    - Original:

$$\min_{\gamma,w,b} \frac{1}{2}\|w\|^2$$
$$\text{s.t. } y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 \quad \forall i \in \{1, \dots, m\}$$

    - ℓ_1 regularization:

$$\min_{\gamma,w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i$$
$$\text{s.t. } y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \xi_i \quad \xi_i \geq 0 \; \forall i \in \{1, \dots, m\}$$

      - $\Rightarrow$ ξ_i permits the data to have a functional margin less than 1.
      - $\Rightarrow$ If the functional margin of a training example is exactly $1 - \xi_i$, the cost function increases by $C\xi_i$.
      - $\Rightarrow$ C balances the goal of minimizing the margins & the goal of ensuring the margins to be at least 1.
  - **Lagrangian for ℓ_1 regularization**:

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2}w^T w + C\sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i\left[y^{(i)}\left(w^T x^{(i)} + b\right) - 1 + \xi_i\right] - \sum_{i=1}^{m} r_i \xi_i$$

- **Dual OP for ℓ_1 regularization**:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)} x^{(j)} \rangle$$

$$\text{s.t.} \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \alpha_i \in [0, C] \quad \forall i \in \{1, \dots, m\}$$

- **KKT Conditions for ℓ_1 regularization**:

$$\begin{aligned} \alpha_i = 0 \quad &\Rightarrow \quad y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 \\ \alpha_i = C \quad &\Rightarrow \quad y^{(i)}\left(w^T x^{(i)} + b\right) \leq 1 \\ 0 < \alpha_i < C \quad &\Rightarrow \quad y^{(i)}\left(w^T x^{(i)} + b\right) = 1 \end{aligned}$$

• SMO [Sequential Minimal Optimization]

- = an efficient algorithm to solve the dual OP above for SVM.
- **Optimization Algorithm 3: Coordinate Ascent**
  - ♦ 1. Gradient Descent & 2. Newton's Method
  - ♦ Problem:

$$\max_{\alpha} W(\alpha_1, \dots \alpha_m)$$

  - ♦ Algorithm:

> Loop until convergence:
>
> For i in range$(m + 1)$:
>
> $$\alpha_i \leftarrow \underset{\widehat{\alpha}_i}{\text{argmax}}\, W(\alpha_1, \dots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \dots, \alpha_m)$$

$\Rightarrow$ Hold other vars fixed & reoptimize w.r.t. one var

- Dual OP of SVM:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)} x^{(j)} \rangle$$

$$\text{s.t.} \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \alpha_i \in [0, C] \quad \forall i \in \{1, \dots, m\}$$

- Coordinate Ascent does not work on this OP:

$$\alpha_i y^{(i)} = -\sum_{j \neq i}^{m} \alpha_j y^{(j)}$$

$$\alpha_i = -y^{(i)} \sum_{j \neq i}^{m} \alpha_j y^{(j)}$$

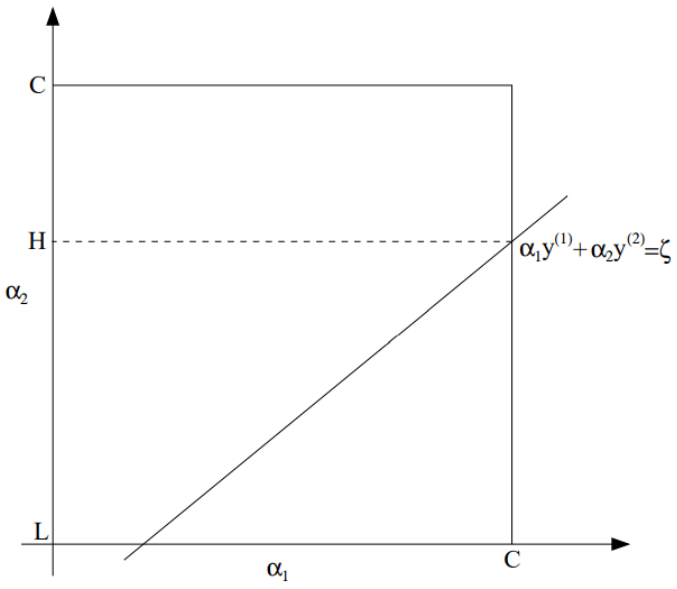
- $y^{(i)} \in \{-1,1\} \ \forall i \in [1, m]$
- As shown, α_i is always determined by all other α_js.
- However, Coordinate Ascent motivates the creation of SMO algorithm, by updating two vars instead of one var at each step.

- Algorithm:

Loop until convergence:
    Select a pair of α_i & α_j # that can maximize the progress
    For i, j in range$(m + 1)$:
        $\alpha_i, \alpha_j \leftarrow \underset{\hat{\alpha}_i, \hat{\alpha}_j}{\operatorname{argmax}} W(\alpha_1, \dots, \hat{\alpha}_i, \dots, \hat{\alpha}_i, \dots, \alpha_m)$

- Visualization:

$$\alpha_j y^{(i)} + \alpha_j y^{(j)} = -\sum_{k \neq i,j}^{m} \alpha_k y^{(k)} = \text{const}$$



- Further Derivation:

$$\because \alpha_i = \left(\mathcal{C} - \alpha_j y^{(j)}\right) y^{(i)}$$

$$\therefore W(\alpha_1, \dots, \alpha_m) = W\left(\dots, \left(\mathcal{C} - \alpha_j y^{(j)}\right) y^{(i)}, \alpha_j, \dots\right) = W\left(\alpha_j\right)$$

$$\because \alpha_i \propto_{\text{linear}} \alpha_j$$

$$\therefore W \propto_{\text{quadratic}} \alpha_j \Rightarrow W = a\alpha_j^2 + b\alpha_j + c$$

$$\therefore \frac{\partial W}{\partial \alpha_j} = 0 \Rightarrow \max W$$

  - Denote this α_j as $\alpha_j^{\text{unclipped}}$, then we can find the optimal α_j in $[L, H]$:

$$\alpha_j^{\text{new}} = \begin{cases} H & \text{if } \alpha_j^{\text{unclipped}} > H \\ \alpha_j^{\text{unclipped}} & \text{if } \alpha_j^{\text{unclipped}} \in [L, H] \\ L & \text{if } \alpha_j^{\text{unclipped}} < L \end{cases}$$

Perceptron

- Problem
  - **Batch learning**: learn from training set → predict on test set
  - **Online learning**: learn & predict on the dataset simultaneously
    - Given a sequence $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$ in order.
    - Given $x^{(1)}$, the algorithm guesses what $y^{(1)}$ is.
    - Given the actual $y^{(1)}$, the algorithm learns & proceeds to the next one.
    - ……

- Model

$$h_w(x) = g(w^T x)$$

where

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

- Algorithm

For i in range$(m + 1)$:
    If $h_w(x) \neq y$:
        $w \leftarrow w + yx$

- **Error Theorem**
  - Given a sequence $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$,
  - If $\|x^{(i)}\| \leq D$ & $\exists u$: $y^{(i)} \cdot (u^T x^{(i)}) \geq \gamma$ (i.e. unit vector u separates data with a margin of at least γ),
  - Then #mistakes that the perceptron makes is at most $\left(\frac{D}{\gamma}\right)^2$.