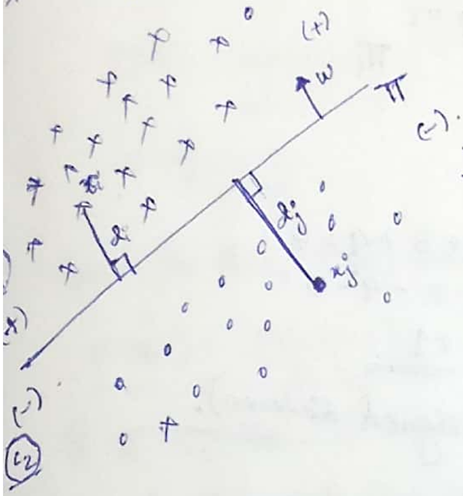
Logistic Regression – (classification Technique).

Geometric Intuition

* Assumption of LR :– classes are almost/perfectly linearly separable.



Given :– $D_n = \{+ve, -ve\}$ pts.

Task :- Find :- w & b $\quad$ $\pi : w^T x + b$

└ find a plane π that best separates +ve pts from negative pts.

say $y_i = +1$ for class 1 $\quad$ $y_i \in \{-1, 1\}$.
& $y_i = -1$ for class 2.

$d_i = \frac{w^T x_i}{\|w\|}$. If w is unit vector, $\|w\| = 1$.
So, $d_i = w^T x_i$.

say $\vec{w}$ points towards the class c_1.
$\vec{w}$ & $\vec{x_i}$ in same dir^n. Hence, $d_i = (+)ve$.
$\vec{w}$ & $\vec{x_j}$ in opposite dir^n. Hence $d_j = (-)ve$.

classifier :→ if $w^T x > 0$
then $y_i = +1$
else
$y_i = -1$

Assume plane passes through origin for simplifying.

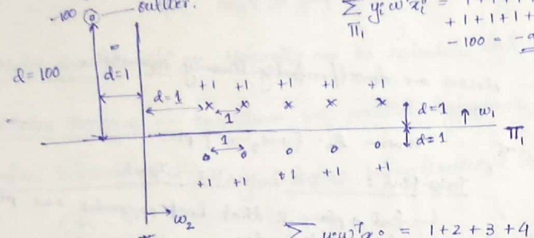
If we consider, $y_i w^T x_i > 0$ for correctly classified pts.
& $y_i w^T x_i < 0$ for mis-classified pts.

For classifier to be good, ⟶ min # misclassifications.
(or)
max # correctly classified pts.

Find a plane π (w, b) s.t. $y_i w^T x_i > 0$ for as many pts as possible.

mathematical optimizat^n fu^n that we need to solve ⟵ $\max_w \sum_{i=1}^{n} y_i w^T x_i$ (Or) $w^* = \underset{w}{\operatorname{argmax}} \left(\sum_{i=1}^{n} y_i w^T x_i \right)$

Squashing & Sigmoid function



$$\sum_{\pi_1} y_i w^T x_i = 1+1+1+1+1+1+1+1+1+1-100 = \underline{-90}$$

$$\sum_{\pi_2} y_i w^T x_i = 1+2+3+4+5-1-2-3-4-5+1 = \underline{+1}$$

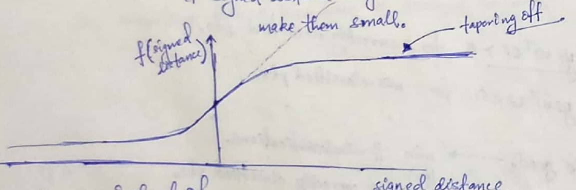
Objective:→ find w that maximizes sum of signed distances. ($y_i w^T x_i$ = signed distance).

So, π_2 is better classifier than π_1.

* But intuitively, π_2 is a terrible classifier. whereas π_1 is a good classifier.

This difference is due to one single extreme outlier point is impacting the model.

Squashing → idea → instead of using signed distances,
if signed distance is small:-
use them as it is.
if signed distance large:-
make them small.

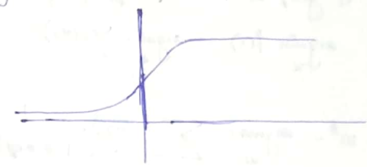


So, now instead of optimizing for

$$\underset{w}{\text{argmax}} \sum_{i=1}^{n} y_i w^T x_i \rightarrow \underset{w}{\text{argmax}} \sum_{i=1}^{n} f(y_i w^T x_i)$$

One such function is called sigmoid function ($\sigma(x)$).

$$\boxed{\sigma(x) = \frac{1}{1+e^{-x}}}$$



Just the shape we wanted.

$\sigma(x)$ max:- 1
min:- 0

$\sigma(0) = 0.5$.

Apart from this, $\sigma(x)$ has a nice probabilistic interpretation.

$\sigma(z_i) = P(y_i = 1)$ $\quad z_i = w^T x_i$

If $z_i = 0$, then it means signed distance = 0.
So, the points lies on the decision boundary. So, $P(y_i=1)$ should be 50%.
that's exactly what $\sigma(z_i)$ gives.

New optimization function → $\boxed{w^* = \underset{w}{\text{argmax}} \sum_{i=1}^{n} \frac{1}{1+\exp(-y_i w^T x_i)}}$
→ less impacted by outliers.

There would be many other functions to do the tapering off. Why σ?
→ ① Nice probabilistic interpretation.
② Easy to differentiate.

Optimization Problem

$$w^* = \underset{w}{\text{argmax}} \sum_{i=1}^{n} \left(\frac{1}{1+\exp(-y_i w^T x_i)} \right) \longrightarrow \text{optimization problem.}$$

Monotonic function → $x\uparrow$, $g(x)\uparrow$ then $g(x)$ is monotonically increasing function.

Ex → $\log(x)$ is monotonically increasing for $x > 0$.

Let $g(x) = \log(x)$.

* $x^* = \underset{x}{\text{argmin}}\, f(x)$. $\quad$ claim $x^* = x'$ as $g(x)$ is monotonic function.
* $x' = \underset{x}{\text{argmin}}\, g(f(x))$

So, if $g(x)$ is monotonic function,

$$\arg\min_x f(x) = \arg\min_x g(f(x))$$

So,

$$w^* = \arg\max_w \sum \log\left\{\frac{1}{1+\exp(-y_i w^T x_i)}\right\}$$

$$w^* = \arg\max_w \sum -\log(1+\exp(-y_i w^T x_i)).$$

$(+1\ \&\ -1)$

$$\boxed{w^* = \arg\min_w \sum \log(1+\exp(-y_i w^T x_i))}$$

$(1\ \&\ 0)$

Alternative formulat^n using probabilistic approach $\rightarrow$

$$\boxed{w^* = \arg\min_w \sum_{i=1}^{n} -y_i \log p_i - (1-y_i)\log(1-p_i)}$$

$$p_i = \sigma(w^T x_i)$$

Both are the same.

Weight Vector

$$w^* = \arg\min_w \sum_{i=1}^{n} \log(1+\exp(-y_i w^T x_i))$$

$\downarrow$

weight vector. $w = \langle w_1, w_2, \ldots, w_d \rangle$ (f_1, f_2, ..., f_d).

decision:- $x_q \rightarrow y_q$ If $w^T x_q > 0$ then $y_q = +1$ else $y_q = -1$

Case I:- (w_i = +ve).

If $f_i \uparrow$ (x_{qi}), $w_i x_{qi} \uparrow \Rightarrow w^T x_q \uparrow \Rightarrow \sigma(w^T x_q)\uparrow \Rightarrow P(y_q = 1)\uparrow$.

Case II: (w_i = -ve)

If $f_i \uparrow$ (x_{qi}), $w_i x_{qi} \downarrow \Rightarrow w^T x_q \downarrow \Rightarrow \sigma(w^T x_q)\downarrow \Rightarrow P(y_q = 1)\downarrow$

$\Downarrow$

$P(y_q = -1)\uparrow$

So, probability of (+)ve cls increases on increasing the i^{th} feature value in query data point if w_i = (+)ve & vice versa.

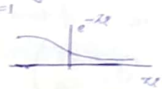
L2 regularization: Overfitting Vs Underfitting

$$w^* = \arg\min_w \sum_{i=1}^{n} \log(1+\exp(-y_i w^T x_i))$$

Let $\boxed{z_i = y_i w^T x_i}$

Then $w^* = \arg\min \sum_{i=1}^{n} \log(1+\exp(-z_i))$

* $\exp(-z_i) \geq 0$



So, $\log(1+\exp(-z_i)) \geq \log(1)$ {Monotonic property}.

$\log(1+\exp(-z_i)) \geq 0$.

Minimal value of $\log(1+\exp(-z_i))$ is '0'.

It occurs when $z_i \rightarrow +\infty$.

So, $\sum_{i=1}^{n} \log(1+\exp(-z_i))$ is 0 when $z_i \rightarrow +\infty$ for all $i \in \{1, 2, \ldots n\}$.

So, we need to adjust w_i's such that all z_i's $\rightarrow \infty$.

such a w is the best w..

But the problem with this is overfitting.

We may overfit to the given dataset while trying to achieve $z_i \rightarrow \infty$ $\forall i \in \{1, \ldots n\}$

* For perfectly fitting each and every point, we may end up having all w_i's as $+\infty$ or $-\infty$

* So, one way we can limit the model from overfitting to given data is by kind of limiting its magnitude.

That's what we do in regularizat^n.

Regularizat^n:

$$w^* = \arg\min_w \underbrace{\sum_{i=1}^{n} \log(1+\exp(-y_i w^T x_i))}_{\text{loss term}} + \underbrace{\lambda w^T w}_{\text{regularizat}^n\text{ term}}$$

Case I: w_j's $\rightarrow +\infty / -\infty$	Case II:- w_j's $\rightarrow$ small.
loss term $\rightarrow 0$ regularizat^n term $\rightarrow \infty$	loss term $\rightarrow$ large regularzn term $\rightarrow 0$.

So, optimizing for both will end up giving as model best fit.

λ — hyper-parameter. — found via CV.

- small → overfitting.
- large → underfitting.

Many optimizatⁿ functions in ML are of form

$$\min \left(\begin{array}{l} \text{loss function} \\ \text{of training data} \end{array} + \text{regularizat}^n \right)$$

$$\boxed{\|w\|_2^2 = w^T w}$$

L1 regularizatⁿ and sparsity

$$w^* = \underset{w}{\operatorname{argmin}} \underbrace{\sum_{i=1}^{n} \log(1+\exp(-y_i w^T x_i))}_{\text{logistic loss}} + \underbrace{\lambda \|w\|_2^2}_{\text{L2-regularizat}^n}$$

* Now, instead of L2-regularizatⁿ, we can also use L1-regularizatⁿ.

$$\boxed{\|w\|_1 = \sum_{i=1}^{d} |w_i|} \text{ — L1-norm.}$$

$$w^* = \underset{w}{\operatorname{argmin}} \left(\begin{array}{l} \text{logistic loss} \\ \text{for training data} \end{array} \right) + \lambda \|w\|_1$$

Advantage of L1-reg → causes sparsity. (in w-vector).

* $w_i = 0$ for all w_i corresponding to less important features.
* In L2-reg w_i's becomes small for less imp. features, but not '0'.

why L1 causes sparsity as compared to L2?

↳ see in optimizatⁿ, intuitive solⁿ.

Elastic-Net :-

$$\boxed{w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} \log(1+\exp(-y_i w^T x_i)) + \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2}$$

Probabilistic Interpretation

For derivation, refer pdf - cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf.

From geometry we get

$$w^* = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \log(1+\exp(-y_i w^T x_i)) + \text{reg.}$$

y_i: +1 or -1.

From probability, after derivatⁿ, we'll get

$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} -y_i \log p_i - (1-y_i) * \log(1-p_i) + \text{reg.}$$

y_i: +1 or 0

where $p_i = \sigma(w^T x_i)$.

We'll see that both are the same only. Ignore the reg. term for the time being.

Case I: $y_i = +ve.$

geom:- $y_i = +1$

prob:- $y_i = +1$.

geom:- $\log(1+\exp(-w^T x_i))$ ✓

prob:- $-1 * \log \dfrac{1}{1+\exp(-w^T x)} = \log(1+\exp(-w^T x))$ ✓

Case II: $y_i = -ve.$

geom:- $y_i = -1$.

prob:- $y_i = 0$.

geom:- $\log(1+\exp(w^T x_i))$

prob:- $-(1-0) \log\left[1 - \dfrac{1}{1+\exp(-w^T x_i)}\right]$

$$= -\log\left[\frac{\cancel{1} + \exp(-w^T x_i) - \cancel{1}}{1+\exp(-w^T x_i)}\right]$$

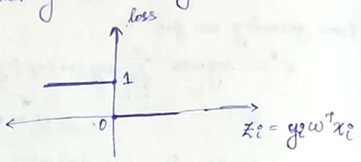
$$= \log\left[\frac{1+\exp(-w^T x_i)}{\exp(-w^T x_i)}\right]$$

$$= \log\left[1 + \frac{1}{\exp(-w^T x_i)}\right] = \log[1+\exp(w^T x_i)]$$

Loss minimization interpretation of LR

An ideal loss-function would be something like following—

- +1 : incorrectly classified pts.
- 0 : correctly classified pts.



Such a loss function is called 0_1 loss function.

$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} 0\_1\ loss\,(x_i, y_i, w)$$

$$0\_1\ loss\,(z_i) = \begin{cases} 1 & \text{if } z_i < 0 \\ 0 & \text{if } z_i \geq 0. \end{cases}$$

But a big issue with selecting this loss funⁿ :- non-differentiable.

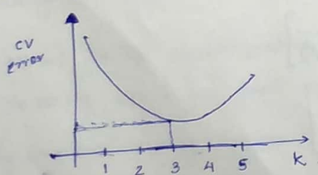
* Since its not differentiable, we try to find a loss function which is close to 0_1 loss but differentiable.

* One such approximation is logistic loss. — gives L.R. (Logistic regression)
Another approximation would be hinge loss. — gives SVM.
Square loss — linear regression.
exp. loss — AdaBoost.

Refer :- Loss functions for classification — Wikipedia for graphs.

Hyperparameter Search / Optimization

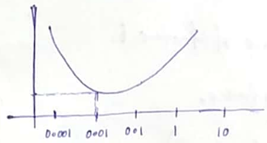
In KNN, we used CV error to determine hyperparameter values.



But k in KNN is integer.
whereas λ in L.R. is a real no.
It can have infinitely many possibilities.
So, we have different techniques for finding values of λ.

① Grid Search

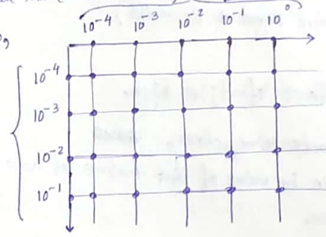
$\lambda = [0.001, 0.01, 0.1, 1, 10, 100]$. By doing the 10x jump, we can search a much larger space.



Grid search is a brute force technique.

If we have elastic-net, m_2 $\lambda_1\|w\|_1 + \lambda_2\|w\|_2$. we have 2 parameters.

So,



for 2 hyper-parameters, $(m_1^n \times m_2)$ values we have to search if we are doing grid search.
So, as #hyperparameter ↑, #times the model needs to be trained increases exponentially.

* In deep learning, we have 10s & 100s of hyperparameters.
So, we have an alternate approach.

② Random Search

$\lambda \in [10^{-4}, 10^{4}]$ — randomly pick values in the given interval.

scientifically proven that random search is as good as grid search and also at times faster than grid search.

Both search available in sci-kit.

Column / feature standardization (Mean centering & scaling).

Just like in K-NN, here also in Logistic Regression, we are dealing with distances from the separating hyperplane.
So, different scales of different features can impact the model.

* So, its mandatory to perform feature standardization before training on D_{train}.

Scanned by CamScanner

Feature importance and Model Interpretability

In KNN, we found feature imp. by forward feature select^n.

f^n. * doubt In NB, using $P(X_i | Y=+1)$ we found importance of feature i.

In LR, we can use w_j's to get feature importance.

Assumption:- All features are independent. (Naive Bayes).

Afterall, L.R. is (Gaussian Naive Bayes + Bernoulli)

If $|w_j|$ is large, then the importance of f_j is high.

If w_j is (+)ve and large, it impacts (+)ve class.
with increase in value of that feature of data point
& vice-versa.

Ex predict gender :- male or female
(+1) (-1)

(hl) hair_length: $|w_{HL}|$ is large & w_{HL} is (-)ve.

$|w_{HL}| \uparrow$; $P(y_q=-1) \uparrow$ as probability of being female inc $\uparrow$ with increase in hair length.

(h) height: $|w_H|$ is large and w_H is (+)ve.

$|w_H| \uparrow$; $P(y_q=+1) \uparrow$

Model Interpretability → Get the most important features & analyze them to give reasoning of the model behaviour.

* Can handpick top 10 features or so and do the above.

Co-linearity and Multi-collinearity

Co-linearity:- f_i & f_j are colinear if

$$\boxed{f_i = \alpha f_j + \beta}$$

Multicollinearity:- if f_1, f_2, f_3, f_4 .. s.t.

$$\boxed{f_1 = \alpha_2 f_2 + \alpha_3 f_3 + \alpha_4 f_4 + \alpha_1}$$

Then f_1, f_2, f_3 & f_4 are said to be multi-colinear.

Q. why does $|w_j|$ not be useful as F.I. if features are co-linear / multilinear?

→ say $w^* = <1, 2, 3>$ $\quad x_q = <x_{q_1}, x_{q_2}, x_{q_3}>$.

Now say we have $f_2 = 1.5 f_1$.

Then, $w^{*T} x_q = 1 \cdot x_{q_1} + 2 \cdot x_{q_2} + 3 x_{q_3}$.

$= x_{q_1} + 2(1.5 x_{q_1}) + 3 x_{q_3}$

$= 4 x_{q_1} + 3 x_{q_3} = <4, 0, 3>$.

So, $<1,2,3>$ & $<4,0,3>$ - both feature vectors give the same result. ~~Say~~

Thus, due to the linear relat^n among features, we can change the weight vector without affecting the result it'll yield.

So, we can't decide the feature importance uniquely.

So, before using w_j as feature importance metrics, we must determine whether features are multi-collinear or not.

Pertubation Technique :→
(Shake a little). For all datapoints of D_{train}, take each x_{ij} & add a little noise to each.

$x_{ij} + \epsilon$ — small noise $N(\mu, \sigma)$
$(0, 0.10)$.

Train model before & after pertubation.

Scanned by CamScanner

before $w = \langle w_1, w_2, \cdots w_d \rangle$

after $\tilde{w} = \langle \tilde{w}_1, \tilde{w}_2, \cdots \tilde{w}_d \rangle$

If w_i & $\tilde{w}_i$ differ significantly, then features are colinear.

⇓

can't use $|w_j|$ as feature importance.

* we always have forward selection technique for feature importance.

Train & Runtime Space & Time Complexity

Train LR:- It is time taken in solving the optimizat^n equation

$w^* = \underset{w}{\text{argmin}}$ (logistic loss + regz^n).

We'll see details in optimization chapter using SGD (stochiastic gradient descent).

It takes $O(nd)$ roughly.

n = # of training pts.

d - dimensionality.

Runtime LR:-

Space complexity → $O(d)$ $\quad w^* = \langle w_1, w_2, \cdots w_d \rangle$.

Time complexity → $O(d)$.

So, if d is small, L.R. is apt. for low latency applications.

Also very memory efficient.

In case of high dimensionality, we can make use of L1 regularizat^n to create sparsity in w so as to reduce calc^{n}s during runtime.

We can adjust λ to create proper amount of sparsity.

If $\lambda \uparrow$, bias ↑.

$\lambda \downarrow$, latency ↑ & variance ↑

So, we have to decide based on our business model & requirements

Real world cases

Decision surface → Linear / hyperplane.

Assumption → data is linearly separable or almost linearly separable.

Imbalanced data → upsampling / downsampling.

Outliers → less impact due to $\sigma(x)$ function tapering off nature.

But not completely avoided.

In order to handle them, we use following filtering technique:-

① $D_{train} \rightarrow w^*$

② $x_i \rightarrow w^{*T} x_i$ - distance of x_i fm π to x_i

③ remove pts which are very far away fm $D_{train} \xrightarrow{to} D'_{train}$ give

④ $D'_{train} \rightarrow \tilde{w}^* \rightarrow$ final sol^n.

Missing values → standard mean / median imputation techniques.

Multi-class :→ One Vs Rest → Typically

extensions $\begin{cases} \text{Softmax} \\ \text{Multinomial LR} \end{cases}$ → deep learning

Similarity Matrix → can be used with an extension LR → Kernel LR

Best and Worst Cases

* almost linearly separable.
* low latency requirement.
* very fast to train.

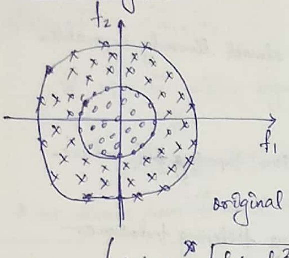
Large d → chances of linearly separable high.

↓

for low latency → L1 reg used.

* LR works decent even upto 1000 dimensions.

Non-linearly separable data

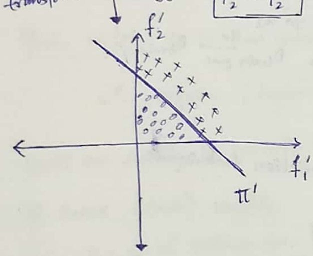


Can we separate the classes using LR?

→ can't separate the two classes by drawing a plane.

Original features $\langle f_1, f_2 \rangle$

feature transform" — feature engg. → $\boxed{f_1' = f_1^2 \quad f_2' = f_2^2}$



In this transformed plane, we can find a hyperplane π' that separates the two classes.

Similarly, when we get a query point x_q,

$x_q = \langle x_{q_1}, x_{q_2} \rangle$
↓ FT or FE
$x_q' = \langle x_{q_1}', x_{q_2}' \rangle \longrightarrow \boxed{\text{LR model}} \longrightarrow y_q$
$\langle x_{q_1}^2, x_{q_2}^2 \rangle$ $\quad \pi'$

But how to come up with feature transforms?

→ comes with practice.

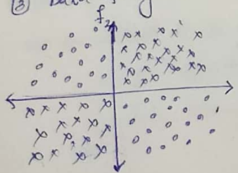
Given data → in form of concentric circles

Eqn of circle → $\boxed{f_1^2 + f_2^2 = r}$ — Eqn not linear in f_1 & f_2, but linear in f_1^2 & f_2^2.

Most important aspect of Applied ML.

1. feature engg ——— (creativity reqd., Art part of ML).
2. Bias-variance tradeoff.
3. Data Analysis.

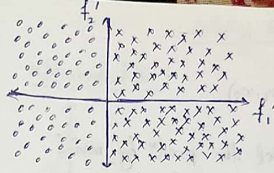
Ex



how to transform to use LR?

→ The dataset is called XOR dataset.

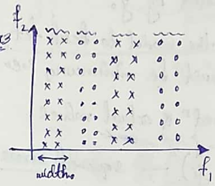
XOR	0	1
0	0	1
1	1	0



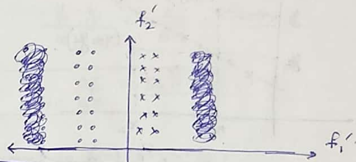
$\boxed{f_1' = f_1 \cdot f_2 \quad f_2' = f_2}$

Now linearly separable.

Ex



feature transform



$\boxed{f_1' = \sin(f_1) \quad f_2' = f_1}$ assuming width is $\pi/2$.

Typical transforms for real value features.

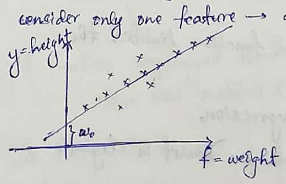
1. $f_1 * f_2,\ f_1^2,\ f_1^3,\ f_1^2 f_2$, etc. — polynomial features.
2. $\sin(f_1),\ \cos(f_1),\ \sin(f_1) * \cos(f_2)$ — trigonometric features.
3. boolean features → OR, AND, XOR.
4. $\log f_i,\ e^{f_i}$.

Linear Regression

└ does actual regression.

Ex → predict height ∈ R given weight, gender, ethnicity, hair color, etc.

consider only one feature → weight for simplicity.



Linear regression → find a line that best fits the given data.

height = w_1 * weight + w_0

$y = mx + c$

In general, we have to predict a hyperplane that best fits the given data.

Scanned by CamScanner