Avaliação de Modelos

Real / **Projected**

	P	N
P	TP	FP
N	FN	TN

Accuracy: $\frac{TP+TN}{All}$

Recall/ Sensitivity: $\frac{TP}{TP+FN}$ Recall = % de P bem previstos entre todos os reais P

Precision: $\frac{TP}{TP+FP}$ Precision = % de P bem previstos entre todos os previstos como P

Fallout/ Specificity: $\frac{TN}{TN+FP}$ Recall ao contrário

F-measure: $\frac{(1+\beta^2)\cdot prec\cdot rec}{(\beta^2\cdot prec)+rec}$

F1-score: $\beta = 1 \rightarrow \frac{2\cdot prec\cdot rec}{prec+rec}$

Error Classification Rate = 1 - accuracy

Teoria da Informação

Entropy: $H(X) = -\sum_{x\in X} p_x \log_2 p_x$

Conditional Entropy: $H(Z \mid X) = \sum_{x\in X} p_x H(Z \mid X = x)$

Information Gain: $IG(Z \mid X) = H(Z) - H(Z \mid X)$

Cross-Entropy: $H(P,Q) = -\sum_{x\in\mathcal{X}} P_x \log Q_x$

Min-Max Scaling: $x_i' = \frac{x_i - \min x}{\max x - \min x}$

Normalization: $x_i' = \frac{x_i - \mu_x}{\sigma_x}$

Surprise: $s_k = \frac{1}{p(x_k)}$

Information: $I_k = \log_2 s_k = -\log_2 p(x_k)$

Funções de Erro - Regressores

SSE: $\frac{1}{2}\sum_{i=1}^{N}(z_i - \hat{z}_i)^2$

MSE: $\frac{1}{N}\sum_{i=1}^{N}(z_i - \hat{z}_i)^2$

RMSE $\sqrt{MSE}$

MAE: $\frac{1}{N}\sum_{i=1}^{N} \mid z_i - \hat{z}_i \mid$

Funções de Erro - Classificadores

Cross-Entropy: $-\sum_{i=1}^{N} z_i \log \hat{z}_i$

Distribuições Normais

$$\sigma = \sqrt{\frac{\sum(x_i-\mu)^2}{N-1}}$$

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

Univariate Distribution: $P(X=x) \sim \mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(\frac{-1}{2\sigma^2}(x-\mu)^2\right)$

Multivariate Distribution: $P(X=x) \sim \mathcal{N}(x \mid \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n \mid \Sigma \mid}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i = \begin{bmatrix}\mu_1\\ \vdots \\ \mu_n\end{bmatrix} \qquad \Sigma = \frac{1}{N-1}\sum_{i=1}^{N}(x_i-\mu)(x_i-\mu)^T = \begin{bmatrix}\sigma_1^2 & \cdots & cov(1,n)\\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \sigma_n^2\end{bmatrix}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{\left(\begin{bmatrix}0\\0\end{bmatrix}+\begin{bmatrix}4\\0\end{bmatrix}+\begin{bmatrix}2\\1\end{bmatrix}+\begin{bmatrix}6\\3\end{bmatrix}\right)}{4} = \begin{bmatrix}3\\1\end{bmatrix}$$

$$\Sigma = \frac{1}{N-1}\sum_{i=1}^{N}(x_i-\mu)(x_i-\mu)^T = \frac{1}{3}\left[\left(\begin{bmatrix}0\\0\end{bmatrix}-\begin{bmatrix}3\\1\end{bmatrix}\right)\left(\begin{bmatrix}0\\0\end{bmatrix}-\begin{bmatrix}3\\1\end{bmatrix}\right)^T+\cdots\right] = \begin{bmatrix}6.66667 & 2.66667\\ 2.66667 & 2\end{bmatrix}$$

Cross Validation

k-fold: dividir data set em k folds, k iterações onde cada fold tem oportunidade de ser train set, ver qual o melhor; performance: avg/stddev entre folds; pode ser **stratified** caso procuremos manter a distribuição de classes igual entre folds

Leave One out é o caso extremo, com fold-size=1

Bayesian Learning

Bayes' Rule: $P(C=k \mid x) = \frac{P(x \mid C=k)P(C=k)}{P(x)}$

Posterior / Likelihood / Prior

MAP Assumption: Escolhemos a classe que maximiza o produto entre Likelihood e Prior.

MLE: Escolhemos a classe que maximiza o Likelihood (elimina o viés do Prior).

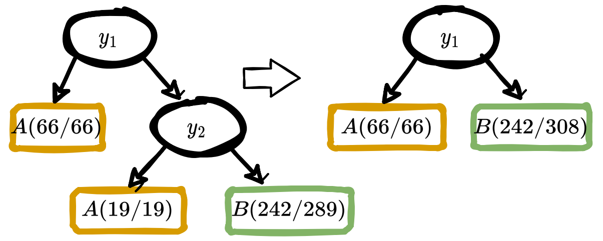
Naive Bayes Assumption: Assumimos independência condicional entre as features

$$P(C \mid x) = \frac{P(C)\prod_{i=1}^{K} P(x_i \mid C)}{\prod_{i=1}^{K} P(x_i)}$$

Árvores de Decisão

Para escolher que feature colocar num dado nível, este ano apenas aprendemos a técnica ID3: escolher a que tem mais IG

Pruning:



Lazy Learning

Manhattan Distance: $\|(x_1,x_2)\|_1 = \sum_{i=1}^{N} \mid x_1^{(i)} - x_2^{(i)} \mid$

Euclidean Distance: $\|(x_1,x_2)\|_2 = \sqrt{\sum_{i=1}^{N}(x_1^{(i)} - x_2^{(i)})^2}$

Cosine Similarities: $\cos(x_1,x_2) = \frac{x_1\cdot x_2}{\|x_1\|\|x_2\|}$

Hamming Distance: quantidade de features que diferem entre 2 samples

Nota: escolhemos sempre os vizinhos com distância menor/com maior semelhança de cossenos - semelhança 1 = vetores sobrepostos, muito próximos

Nota: um "leave one out schema" refere-se a podermos escolher qualquer um dos outros pontos do data set para vizinho, exceto ele próprio

KNN Algorithm:

1: Calcular distâncias a todos os vizinhos, escolher os k mais próximos.

2: Realizar a previsão: moda no caso da classificação, média no caso da regressão

Podemos ainda pesar as nossas escolhas: $d(x_1,x_2) = |1-2|+|1-1| = 1$

$d(x_1,x_3) = 3, \quad d(x_1,x_4) = 4, \quad d(x_1,x_5) = 2, \quad d(x_1,x_6) = 1$

$\hat{z}_1 = \textbf{weighted\_mode}(1/1B, (1/1+1/2)A) = A$

$\hat{z}_2 = \textbf{weighted\_mean} = \frac{1/1\cdot 0.5 + 1/2\cdot 0.7 + 1/1\cdot 1.2}{1/1+1/2+1/1} = 0.82$

kNN pode ter problemas com distâncias Euclideanas, dando mais importância a variáveis com maior "range" de valores - evitamos isso normalizando/escalando

$$\begin{bmatrix}1\\2\\3\\4\\5\end{bmatrix}\begin{bmatrix}1\\2\\3\\4\\5\end{bmatrix}^T = \begin{bmatrix}1&2&3&4&5\\2&4&6&8&10\\3&6&9&12&15\\4&8&12&16&20\\5&10&15&20&25\end{bmatrix}$$

$$\begin{bmatrix}a_1&b_1\\c_1&d_1\end{bmatrix}\cdot\begin{bmatrix}a_2&b_2\\c_2&d_2\end{bmatrix} \downarrow \begin{bmatrix}a_1a_2+b_1c_2 & a_1b_2+b_1d_2\\ c_1a_2+d_1c_2 & c_1b_2+d_1d_2\end{bmatrix}$$

Regressão Linear/NL

Nota: matriz X tem as samples 1 por linha, i.e x_10 está na primeira linha, primeira coluna sendo a coordenada 0 (=1) de x_1

Hyperplane: $z = \sum_{i=0}^{N} w_i x_i = w^T x, \quad x_0 = 1$

Caso não seja uma RL: $x \overset{\phi(x)}{\mapsto} \Phi = \begin{bmatrix}1 & \phi_1(x_1) & \cdots\\ 1 & \phi_1(x_2) & \cdots\\ \vdots & \vdots & \vdots\\ 1 & \phi_1(x_n) & \cdots\end{bmatrix}$

Weight Update (normal): $w = (X^TX)^{-1}X^TZ$ or $w = (\Phi^T\Phi)^{-1}\Phi^TZ$

Ridge Regression: $E(w) = \frac{1}{2}\sum(z_i-\hat{z}_i)^2 + \lambda\sum_{i=0}^{m} w_i^2 \nearrow \lambda\|w\|_2^2$

o que estamos a fazer é resolver a regressão via gradient descent, com error function E(w), para ambas

$w = (X^TX+\lambda I)^{-1}X^TZ$

Lasso Regression: E(w) igual a Ridge mas a norma não é ao quadrado e é a L1 norm

λ **: Termo de regularização - evita overfitting, estamos a minimizar a SSE**

The Ridge regularization technique aims to **tune** a linear regression model, by adding a penalty term to the SSE. This penalty, usually denoted by λ, changes the closed form solution for the weights, W, to the following:

Ridge = L2 regularization

$$W = (X^TX+\lambda I)^{-1}X^TZ$$

where I is the identity matrix. The regularization term, λ, is a hyperparameter that controls the amount of regularization applied to the model, in order to avoid overfitting. The larger the value of λ, the more the regression will shift from its original solution.

LASSO = L1 regularization

LASSO, or Least Absolute Shrinkage and Selection Operator, is another regularization technique that aims to tune a linear regression model. It differs from Ridge regression in that it adds a penalty term to the SSE, but this term is the sum of the absolute values of the weights, instead of the sum of their squares. With LASSO, the coefficients are shrunk towards zero, which means that some of them will be reduced to zero, effectively removing them from the model - this is a so-called **feature selection**, since it'll happen to the least relevant features of the model. Ridge regression, on the other hand, will only reduce the weights' magnitude, but will never set them to zero. This way, LASSO is more likely to produce a sparse (and better) model, with fewer features, than Ridge regression, regarding data sets with a large number of features.

Perceptron

O algoritmo do perceptrão baseia-se numa learning rule, que dá update aos pesos a cada sample/batch, e só depois de uma época completa sem alteração ao peso é que podemos dizer que houve convergência.

Nota: no perceptrão c/ GD, X tem 1 sample por coluna, não/linha

$w_{new} = w_{old} + \eta(z_i - \hat{z}_i)\cdot x$

Algoritmo default

Output do perceptrão: $\hat{z}(x) = f(net(x)), \quad net(x) = w^Tx$

Nota: não esquecer de incluir o $x_0 = 1$

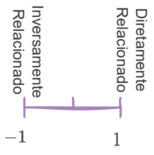
$$-\frac{\partial\sum_{i=1}^{N} z_i\log\hat{z}_i}{\partial w} = -\sum_{i=1}^{N}\frac{z_i\frac{\partial\hat{z}}{\partial w}}{\hat{z}}$$

Derivada cross entropy normal:

Gradient Descent

$w_{new} = w_{old} - \eta\frac{\partial E}{\partial w}$

Pearson Correlation: $\frac{cov(x_1,x_2)}{var(x_1)\cdot var(x_2)}$



$cov(x,y) = \frac{1}{n-1}\sum(x_i-\overline{x})(y_i-\overline{y})$

Spearman Ranking: First, rank the variables (bigger ranks = larger variable values)

Spearman Correlation: After applying Spearman ranking to variables, apply Pearson

$$X = \begin{bmatrix}a & b\\ c & d\end{bmatrix} \qquad X^{-1} = \frac{1}{ad-bc}\begin{bmatrix}d & -b\\ -c & a\end{bmatrix}$$

$$\log_b(a) = \frac{\log_x(a)}{\log_x(b)}$$

Clustering

(vamos trabalhar, aqui, com unsupervised learning)

K-means: (existe um k-medians, onde reutilizamos a lógica mas o update é com medianas)

1: Calcular a distância de cada sample a cada cluster, dar assign de cada sample ao cluster mais próximo.

2: Atualizar os centróides de cada cluster: fazer a média das coordenadas de cada sample, para cada cluster, e as coordenadas dos centróides passam a ser as médias.

Paramos caso tenhamos convergido/a alteração tenha sido muito pequena.

EM: Nota: em k-means faz-se assign de samples a clusters, aqui não!! Nota 2: cada cluster é definido por uma distribuição, aqui o que fazemos é atualizar os parâmetros da mesma.

E-Step: Para cada cluster, calcular o posterior normalizado (gamma) para cada sample (usar a Bayes' rule) **NOTA**: p(x) será a soma de todos os priors * likelihoods daquele cluster

$$\gamma_{ni} = P(c = k_i \mid x_n) = \frac{P(x_n \mid c = k_i)\pi_i}{\sum_{j=1}^{k} P(x_n \mid c = k_j)\pi_j}$$

M-Step: Atualizar média, variância e priors; usar a média nova no cálculo da variância nova

$$\mu_k = \frac{\sum_{n=1}^N \gamma_{nk} x_n}{\sum_{n=1}^N \gamma_{nk}}, \quad \sigma_k = \sqrt{\frac{\sum_{n=1}^N \gamma_{nk}(x_n - \mu_k)^2}{\sum_{n=1}^N \gamma_{nk}}}, \quad \pi_k = \frac{1}{N}\sum_{n=1}^{N} \gamma_{nk}$$

$$\Sigma_k = \frac{1}{\sum_{n=1}^N \gamma_{nk}} \sum_i \gamma_{ik} \cdot (x_i - \mu_k)(x_i - \mu_k)^T$$

Caso as distribuições fornecidas não sejam gaussianas, os parâmetros a atualizar são likelihoods e priors - priors da mesma maneira, likelihoods como está abaixo

$$P(y_n \mid c = k_i) = \frac{\sum_{j=1}^N \gamma_{ji} x_{jn}}{\sum_{j=1}^N \gamma_{ji}}$$

$$ECR = \frac{FP + FN}{All} \qquad Purity = \frac{1}{N}\sum_{i=1}^{k} \max_j (c_i \cap t_j) = 1 - ECR$$

$$\text{Separation}(k) = \frac{1}{k^2}\sum_{i=1}^{k}\sum_{j=1}^{k} \|\mu_i - \mu_j\|^2 \qquad \text{Cohesion}(k) = \sum_{i=1}^{k}\sum_{x \in k_i} \|x - \mu_i\|^2$$

MLP/NN's

Bias: # colunas = 1, # linhas = nodes layer L

Weights: #colunas = nodes layer L-1, # linhas = nodes layer L

Forward Prop:

$$z^{[\ell]} = w^{[\ell]} x^{[\ell-1]} + b^{[\ell]}$$
$$x^{[\ell]} = f(z^{[\ell]})$$

Back Prop:

SSE $$\delta^{[L]} = \frac{\partial E}{\partial x^{[L]}} \circ \frac{\partial x^{[L]}}{\partial z^{[L]}}$$

CE **E** Softmax $$\delta^{[L]} = \frac{\partial E}{\partial z^{[L]}} = x^{[L]} - z$$

$$\delta^{[\ell]} = \left(\frac{\partial z^{[\ell+1]}}{\partial x^{[\ell]}}\right)^T \delta^{[\ell+1]} \circ \frac{\partial x^{[\ell]}}{\partial z^{[\ell]}}$$

Derivadas

Nota: o $z^{[L]}$ é vulgarmente chamado net

$$\frac{\partial z^{[\ell]}}{\partial x^{[\ell-1]}} = w^{[\ell]}$$
$$\frac{\partial z^{[\ell]}}{\partial w^{[\ell]}} = x^{[\ell-1]}$$
$$\frac{\partial z^{[\ell]}}{\partial b^{[\ell]}} = 1$$
$$\frac{\partial x^{[\ell]}}{\partial z^{[\ell]}} = \frac{\partial}{\partial z^{[\ell]}} f(z^{[\ell]})$$
$$\frac{\partial E}{\partial w^{[\ell]}} = \delta^{[\ell]} (x^{[\ell-1]})^T$$

Update Step: Atualizar weights e biases $\rightarrow$ $$\frac{\partial E}{\partial b^{[\ell]}} = \delta^{[\ell]} \left(\frac{\partial z^{[\ell]}}{\partial b^{[\ell]}}\right)^T = \delta[\ell]$$

Stochastic: Fazer os updates para uma única sample $$w^{[\ell]} = w^{[\ell]} - \eta \frac{\partial E}{\partial w^{[\ell]}}$$

Batch: Fazer os updates para um conjunto de samples, pode-se combinar tudo e ter uma coluna/sample nos nets e layer outputs $$b^{[\ell]} = b^{[\ell]} - \eta \frac{\partial E}{\partial b^{[\ell]}}$$

Funções de Ativação:

$$\tanh(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1} \qquad \frac{\partial \tanh(\alpha z^{[\ell]})}{\partial z^{[\ell]}} = \alpha(1 - \tanh(\alpha z^{[\ell]})^2)$$

$$\sigma(\alpha x) = \frac{1}{1 + \exp(-\alpha x)} \qquad \text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^n \exp(z_j)}, \quad \frac{\partial s(z_i)}{\partial z_j} = \begin{cases} s(z_i)\cdot(1 - s(z_i)), & i = j \\ -s(z_i)\cdot s(z_j), & i \neq j \end{cases}$$

$$\frac{\partial \sigma(\alpha x)}{\partial w} = \alpha\sigma(\alpha x)(1 - \sigma(\alpha x)) \qquad ReLU(x) = \max(0, x)$$

The derivative is:

assumimos 0 para x=0 $$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

ReLU: ReLU funciona bem para evitar vanishing gradients (i.e quando os updates começam a não fazer quase nada)

Early Stopping funciona bem para evitar overfitting: paramos quando um dado validation score não fica acima de X durante Y épocas

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

se as funções de ativação não tiverem exclusivamente valores entre 0 e 1, não devemos usá-las como ativação em gradient descent com CE, porque não representarão probabilidades

$$\text{step}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Btw the silhouette of the cluster is the average of its samples' silhouettes, where -1 is bad, 0 is moderate and 1 is amazing (score's logic is the same for both clusters and samples in that regard)

$$s(x_n) = \frac{b_n - a_n}{max\{a_n, b_n\}}$$

$$a_n = \frac{1}{\underset{x_n \in c_k}{|c_k| - 1}} \sum_{x_i \in c_k, x_i \neq x_n} \|x_i - x_n\|$$

a_n corresponde à distância média de uma sample a todas as outras sample do seu centroide

$$b_n = \min_{n' \in \{1,\cdots,N\}, n' \neq k} \frac{1}{|c_{n'}|} \sum_{x_i \in c_{n'}} \|x_i - x_n\|$$
$b_n \in c_k$

b_n corresponde à distância média de uma sample a todas as outras do seu cluster vizinho, onde o cluster vizinho é o que tem as samples que minimizam esta média

DATA PROBABILITY

$$P(X) = \prod_{n=1}^{N} P(x_n)$$

$$P(X_n) = \sum_{i=1}^{K}\prod_{j=1}^{M} P(c = k_i) P(x_{nj} \mid c = k_i)$$

Data probability: a probabilidade das nossas distribution mixtures produzirem o data set; Sinónimo de **fitting probability**

Dimensionality Reduction

Objetivo: reduzir o #variáveis através de uma transform. linear, preservando o máximo de informação possível.

PCA: decompõe o espaço de features em componentes principais - PC1 é a direção mais informativa, com maior variância associada (maior valor próprio), PCA2 a segunda, ortogonal à primeira, etc. No fundo, o PCA vai projetar os dados ao longo dos eixos que preservam maior variabilidade.

Se queremos preservar uma proporção p da variabilidade, vamos precisar de manter k das n features:

$$p = \frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^n \lambda_i}$$

Critério de Kaiser: podemos descartar features com valor próprio menor que 1

Reconstruction Error: $$\frac{1}{2}\sum_{i=k+1}^{m} \lambda_i$$

K-L transform:

- Calcular média e matriz Sigma, se for caso disso;
- Calcular valores próprios de Sigma, det(Sigma - lambda I)
- Calcular vetores próprios: $\Sigma v_k = \lambda_k v_k$
- Fazer matrix K-L, onde o primeiro vetor é o do menor lambda, etc

Se quisermos obter o angulo de rotação, fazer $$cos(v_k, e_k) = \frac{v_k \cdot e_k}{\|v_k\|\|e_k\|}$$

Encontrar novos pontos (projetados no novo sistema de eixos): multiplicar matriz K-L pelos pontos (transform. linear): $$x' = U_{K-L}^T x$$

---

a) i) $$(c-1) + c\cdot(n^m - 1)$$

a) ii) $$(c-1) + c\cdot(m\cdot(n-1))$$

b) i) $$(c-1) + c\cdot\left(m + m\cdot\frac{m+1}{2}\right)$$

b) ii) $$(c-1) + c\cdot 2m$$

Nota: multivariate, Sigma é simétrica logo só se calcula metade da matriz; univariada é calcular 1 sigma por feature
Nota 2: (c-1) refere-se aos priors, o resto likelihoods

4. Assuming training examples with m features and a binary class:
   (a) How many parameters are needed to estimate, considering boolean features and:
      i. No assumptions regarding the data's distribution.
      ii. Naive Bayes assumption.
   (b) How many parameters are needed to estimate, considering numeric-valued features and:
      i. Multivariate Gaussian assumption.
      ii. Naive Bayes with a Gaussian assumption.

c = #classes, m = #features
n = #values por feature

Consider an MLP with three layers, architecture $[d, n, m]$. How many parameters will we need to learn?

We'll need to learn weights and biases: the amount of parameters is the sum of the "products" below

$$W^{[1]} = n \times d, b^{[1]} = n \times 1, W^{[2]} = m \times n, b^{[2]} = m \times 1$$

Note: models are in more risk to overfit if they have a higher VC dimension (can be approx. by #parameters)

How many parameters will we need to learn in the conditions below?

i. a fixed feature transformation $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^6$ followed by a perceptron.
ii. a learnable feature transformation $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^2$ that depends on 4 parameters, followed by a perceptron

In model 1 the feature transformation is fixed, so it contributes with no parameters. The perceptron is applied after the transformation to 6 dimensional inputs. So in total, 6+1=7 parameters.
In model 2 the transformation requires the learning of 4 parameters. The perceptron is applied after the transformation to 2 dimensional inputs, i.e. 2+1=3 parameters. Adding both parts: 4+3=7 parameters.

Random forests: ensemble learning methods for classification and regression; we build a multitude of decision trees at training time: for classification, we pick, for a given sample, the class most picked by the trees; for regression tasks, we pick the average prediction of all trees; for high-dimensional data, they avoid overfitting present in regular decision tree approaches.
Individual Decision trees are prone to overfitting, especially when a tree is particularly deep. This is due to the amount of specificity we look at leading to smaller sample of events that meet the previous assumptions. This small sample could lead to unsound conclusions.

Finally, let's talk about the **random classifier**. A random classifier is a classifier that makes random predictions, and is usually used as a baseline for comparison with other classifiers. In this case, we have three classes, A, B and C, therefore the random classifier will predict each class with probability $\frac{1}{3}$. As such, the accuracy of the random classifier is also $\frac{1}{3}$.

Regarding both sensitivity and precision, we have the following (considering X as the class predicted by the random classifier, $\overline{X}$ as all the remaining classes, and $p = P(X)$):

$$\text{recall}(X) = \frac{TX}{TX + F\overline{X}} = \frac{p\cdot\#X}{p\cdot\#X + (1-p)\#X} = p, \quad \text{precision}(X) = \frac{TX}{TX + FX} = \frac{p\cdot\#X}{p\cdot\#X + p\#\overline{X}} = \frac{\#X}{\#X + \#\overline{X}}$$

What are the maximum likelihood parameters of a multivariate Gaussian distribution for this data set? $\rightarrow$ Encontrar Sigma, média, Sigma^1 e det(Sigma)

What is the problem of working without assumptions?

(this is regarding using Bayes' rule to compute posteriors without any assumptions - i.e, likelihoods aren't assumed to be following naive bayes/gaussian stuff)

Without assumptions, we're essentially strapped to the data set we have at hand. In cases like the one at hand, for example, we're unable to compute the posterior probabilities for the query vector x_{new} (and for a myriad of other query vectors). As such, data sets with a large number of features and/or a small number of samples are problematic to work with without making any assumptions, since we're more likely than not going to run into this problem often.

below: how output thresholds work in classifiers

Output thresholds are used to convert the output of a linear regression model into a class label - considering binary labels, the threshold θ is used to determine whether the output is 0 or 1 (considering whether or not $\hat{z}$ is greater than θ). In this case, we can compute the output of the model for the new sample, x_{new}, as follows:

(se fosse usando uma MLE approach seria default closed form solution, assim é com MAP assumption)

$$z = w_1 y_1 + w_2 y_2 + \epsilon, \epsilon \sim \mathcal{N}(0, 0.1)$$

(b) w using the Bayesian approach, assuming $p(w) = N(w \mid \mu = [0,0], \Sigma = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix})$.

$$\begin{aligned} \text{argmax}_w\, p(w \mid X, Z) &= \text{argmax}_w\, p(X, Z \mid w)p(w) \\ &= \text{argmax}_w\, \Pi_{i=1}^N p(z_i \mid x_i, w)p(w) \end{aligned} \qquad w = (X^T X + \tau\Sigma^{-1})^{-1} X^T Z$$

$$= \text{argmax}_w\, \frac{1}{\tau}\cdot(Z - Xw)^T(Z - Xw) + w^T\Sigma^{-1}w \qquad \lambda = \frac{\tau}{\sigma^2}$$

The softmax activation function aims to scale numbers into probabilities: it's usually present in the output layer of a NN, transforming the last layer's nets into a vector of probabilities.
Even though the sigmoid's plot is, generally speaking, similar to the softmax's one, the two functions are not the same: for a given layer, the sum of the softmax's outputs always sums to 1, while that isn't necessarily the case for the sigmoid

(b) Assuming EM clustering is applied to model all scenarios, what would the means and variances look like? For simplicity, assume that the covariance matrix is diagonal.

Regarding the second part of the question, the means will generally be wherever the data's distributions are centered, while the covariance matrices will be diagonal, with the diagonal values being the variances of each dimension - considering ellipse-like clusters, for example, the variance will be obviously higher in the "horizontal" (i.e the "first" in Σ) dimension, while it will be lower in the "vertical" (i.e the "second" in Σ) dimension (vice-versa for vertically elongated ellipses). Considering circle-like clusters, the variances will generally be the same in both dimensions, with more spread out data points having higher associated variances.

1. Considering $\lambda = 7.877$, finding its eigenvector (with $\Sigma = \begin{bmatrix} 6.667 & 2.667 \\ 2.667 & 2 \end{bmatrix}$):

$$\left[\begin{array}{cc|c} 6.667 - 7.877 & 2.667 & 0 \\ 2.667 & 2 - 7.877 & 0 \end{array}\right] \rightarrow \left[\begin{array}{cc|c} -1.21 & 2.667 & 0 \\ 2.667 & -5.877 & 0 \end{array}\right] \rightarrow \left[\begin{array}{cc|c} 1 & -2.204 & 0 \\ 1 & -2.204 & 0 \end{array}\right] \rightarrow \left[\begin{array}{cc|c} 1 & -2.204 & 0 \\ 0 & 0 & 0 \end{array}\right]$$

From the equation above, we can gather the following eigenvector (normalized to have unit norm):

$$\vec{v} = \begin{bmatrix} 2.204 \\ 1 \end{bmatrix}, \quad \vec{v} = \frac{\vec{v}}{\|\vec{v}\|} = \begin{bmatrix} 0.911 \\ 0.413 \end{bmatrix}$$

(this below is coming from a 2-dimensional Gaussian scenario), productory is because of NB

Compute the most probable class for the query vector, under the Naive Bayes assumption, using 1-dimensional Gaussians to model the likelihoods.

Modelling the likelihoods as 1-dimensional Gaussians (instead of the 2-dimensional ones utilized in the previous exercise) ends up being a simplification of the problem; we can reutilize the previously computed mean vectors to compute the likelihoods for each class, as follows:

$$\mu_0 = \begin{bmatrix} \mu_0^{(1)} \\ \mu_0^{(2)} \end{bmatrix} = \begin{bmatrix} 93.3333 \\ 156.667 \end{bmatrix}, \quad \mu_1 = \begin{bmatrix} \mu_1^{(1)} \\ \mu_1^{(2)} \end{bmatrix} = \begin{bmatrix} 80 \\ 203.333 \end{bmatrix}$$

$$\sigma_0 = \begin{bmatrix} \sqrt{\Sigma_0^{(1,1)}} \\ \sqrt{\Sigma_0^{(2,2)}} \end{bmatrix} = \begin{bmatrix} 66.5833 \\ 5.7735 \end{bmatrix}, \quad \sigma_1 = \begin{bmatrix} \sqrt{\Sigma_1^{(1,1)}} \\ \sqrt{\Sigma_1^{(2,2)}} \end{bmatrix} = \begin{bmatrix} 10 \\ 15.2753 \end{bmatrix}$$

$$P(x \mid z = c) \sim \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu_c}, \boldsymbol{\sigma_c}) = \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi\sigma_{c,i}^2}} \exp\left(-\frac{1}{2\sigma_{c,i}^2}(x_i - \mu_{c,i})^2\right)$$

5. Consider the sum squared and cross-entropy loss functions. Any stands out? What changes when one changes the loss function?

The cross-entropy loss function stands out as it corresponds to making a MLE of the parameter w under the assumption that the data is generated by a Bernoulli distribution, such that the outputs z are sampled from a random variable Z with probability $p(Z = 1) = \sigma(wx)$ and $p(Z = 0) = 1 - \sigma(wx)$. As such, the cross-entropy loss function is the negative log-likelihood of the data under such distribution.

The sum of squares loss function would be more fitted to a regression scenario, since it matches a MLE estimation of w under the assumption that the outputs z are sampled from a random variable $Z \sim \mathcal{N}(wx, \sigma^2)$. Nevertheless, one can use the sum of squares loss function on classification problems as well - as a matter of fact, this function is also convex, thus the use of gradient descent will also find a global minimum. In this case, the difference between the functions will be the value of the minimal parameter: since the functions are different, most likely their minimal values will be different as well, leading to different estimations of the parameter w; regarding classification problems, like the one in hands, the cross-entropy loss function should generally lead to a better estimation of the parameter w.

Input: 100 pixels with real values, each sample can be one of 26 classes
How many trainable parameters if the sample is according to gaussian NB?

There will be 26 priors (one per class), 26*100 likelihoods (one per feature, per class), each likelihood with a mean and a std.dev: (26-1) + 26*100*2