AULA 3
CONJUNTO DE DADOS DESBALANCEADOS

1. Objetivos

- Apresentar normalização de dados.
- Apresentar alguns tipos de métricas.
- Conjunto de dados desbalanceados.

1. Normalização dos dados

- Em geral não é adequado ter dados de entrada para uma RNA com valores absolutos grandes, ou dados que são muito heterogêneos, como por exemplo, algumas características das entradas terem valores entre 0 e 1, e outras entre 100 e 1000.
- Ter dados com valores muito grandes, ou muito heterogêneos, pode gerar problema saturação das funções de ativação e, consequentemente, problemas de "vanishing gradients" e de "exploding gradients".
- Para tornar o treinamento mais eficiente e mais rápido, evitando problemas de "vanishing gradientes" e/ou "exploding gradients", os dados de entrada (e em geral os de saída também) devem ter as seguintes características:
  - Possuírem valores pequenos $\Rightarrow$ tipicamente entre 0 e 1, ou entre -1 e 1, dependendo do tipo de problema;
  - Serem homogêneos $\Rightarrow$ todos os elementos dos vetores de entrada devem possuir o mesmo intervalo de variação.
- É uma prática comum, mas nem sempre necessária, normalizar os exemplos de treinamento de forma que cada elemento de todos os exemplos de treinamento apresente:
  - média igual a zero;
  - variância igual a um.
- Essa normalização é obtida aplicando aos exemplos de treinamento as seguintes operações:

$$\boxed{\overline{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)}} \qquad (1)$$

$$\boldsymbol{\sigma}^2 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}^{(i)} - \overline{\mathbf{x}})^2 \tag{2}$$

$$\mathbf{x}^{(i)} = \frac{\mathbf{x}^{(i)} - \overline{\mathbf{x}}}{\boldsymbol{\sigma}} \tag{3}$$

onde:

$\overline{\mathbf{x}} = \begin{bmatrix} \overline{x}_1 \\ \vdots \\ \overline{x}_{n_x} \end{bmatrix}_{(n_x,1)}$ $\Rightarrow$ vetor com as médias de cada componente dos dados de entrada.

$\boldsymbol{\sigma}^2 = \begin{bmatrix} \sigma_1^2 \\ \vdots \\ \sigma_{n_x}^2 \end{bmatrix}_{(n_x,1)}$ $\Rightarrow$ vetor com as variâncias de cada componente dos dados de entrada.

- **Importante: os vetores de médias e variâncias usados para normalização são calculados usando-se somente os dados de treinamento. Nunca deve usar os dados de validação e teste para calcular os parâmetros usados para fazer a normalização.**

- Note que:
  - A normalização é realizada independentemente para cada elemento dos dados de entrada;
  - A equação (3) representa uma divisão elemento por elemento, onde cada elemento dos vetores ($\mathbf{x}^{(i)} - \overline{\mathbf{x}}$) é dividido pela sua respectiva variância.

- A Figura 1 apresenta um exemplo de dados não normalizados e após a sua normalização para um caso onde os vetores de entrada têm dimensão (2, 1), ou seja, $n_x = 2$.

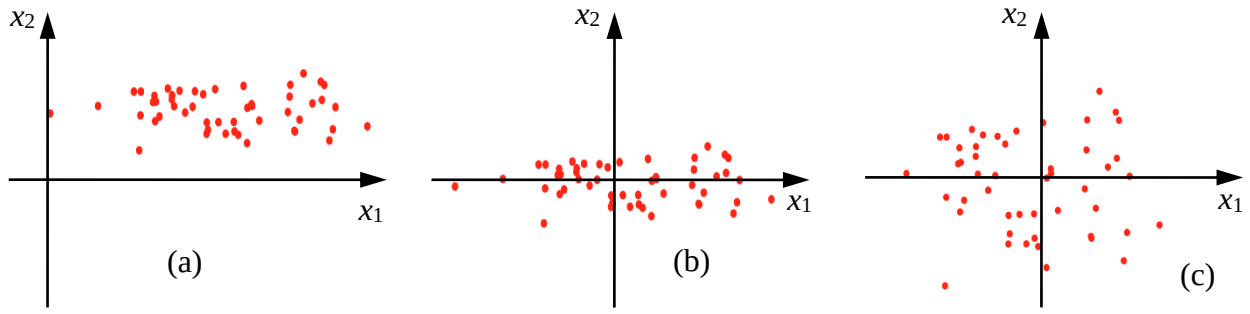

Figura 1. Processo de normalização dos dados de entrada. (a) dados originais; (b) dados com média zero; e (c) dados normalizados (adaptado de Andrew Ng, deeplearning.ai).

- No código a seguir é mostrado um exemplo de como fazer a normalização dos dados de entrada e de saída usando funções da biblioteca numpy.

```
# Cálculo da média e desvio padrão dos dados de entrada de
treinamento
meanx = x_train.mean(axis=0)
stdx = x_train.std(axis=0)

# Normalização das entradas dos dados de treinamento, validação e
teste usando as médias meanx e os desvios padrão stdx
x_train_norm = (x_train - meanx)/stdx
x_val_norm = (x_val - meanx)/stdx
x_test_norm = (x_test - meanx)/stdx

# Cálculo da média e desvio padrão dos dados de saída de treinamento
meany = y_train.mean(axis=0)
stdy = y_train.std(axis=0)

# Normalização das saídas dos dados de treinamento, validação e
teste usando as médias meany e os desvios padrão stdy.
y_train_norm = (y_train - meany)/stdy
y_val_norm = y_val - meany)/stdy
y_test_norm = (y_test - meany)/stdy
```

- Nesse código, as entradas e as saídas de cada exemplo de treinamento são vetores linha.
- As funções `mean` e `std` calculam a média e o desvio padrão de tensores numpy.

- O tratamento dos dados de saídas em geral segue formatos diferentes, dependendo do tipo de problema.

- A grande vantagem de se normalizar os dados é obter uma função de custo que tende a ter um formato mais circular, o que facilita muito a convergência do gradiente descendente. Esse aspecto é mostrado esquematicamente na Figura 2 para um caso hipotético de uma RNA com somente um peso de ligação e um viés.

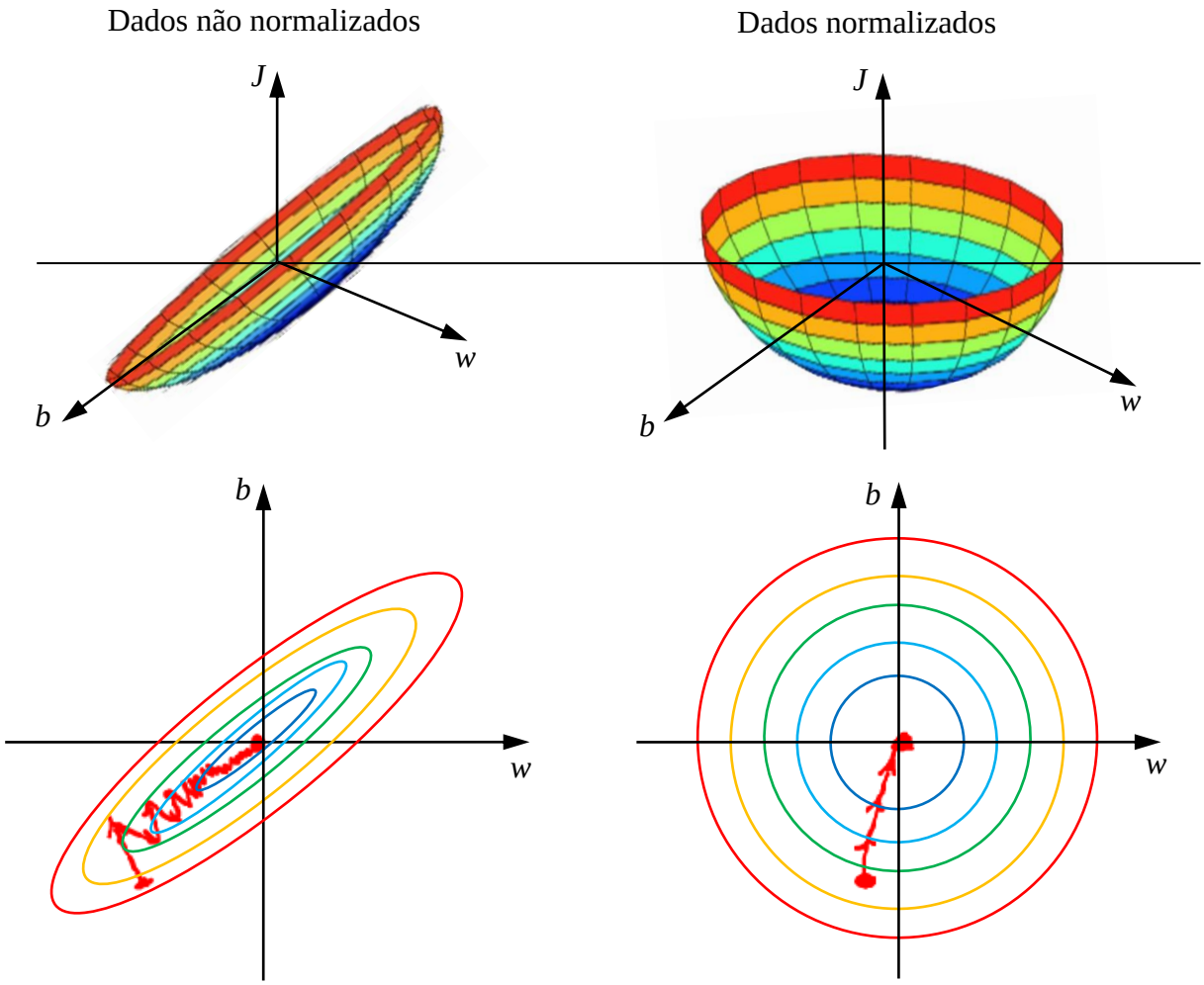

Figura 2. Função de custo *J* com dados não normalizados e com dados normalizados (adaptado de Andrew Ng, deeplearning.ai).

- Da Figura 2, observa-se que o processo de convergência do cálculo dos parâmetros da RNA, com o gradiente descendente é mais eficiente com os dados normalizados.

2. Métricas para medir a eficiência da RNA

- Para controlar o treinamento e verificar a eficiência de uma RNA temos que saber o que queremos alcançar.
- Métricas são funções usadas para medir o desempenho das RNAs.
- Algumas métricas mais usuais são:
  - Exatidão;
  - Erro absoluto médio;

- Precisão e revocação (“precison/recal”);
- Pontuação F1 (“F1 score”);
- Outras.

➢ A métrica não precisa ser igual à função de erro ou função de custo utilizada para treinar a RNA ⇒ mas a escolha da métrica está relacionada com a escolha da função de custo, ou seja, o que se deseja otimizar durante o treinamento da RNA.

2.1 Exatidão

➢ A exatidão de uma solução representa o quão próxima ela está da solução correta.

➢ A forma de calcular a exatidão depende do tipo de problema.

➢ **Para um problema de classificação binária a exatidão (*acc*) pode ser calculada da seguinte forma:**

$$acc = 1 - \frac{1}{m}\sum_{i=1}^{m}\left|c^{(i)} - y^{(i)}\right| \tag{4}$$

onde:

m = número de exemplos;

c = classe prevista pela RNA (igual a 0 ou 1);

y = saída ou classe real (igual a 0 ou 1).

- A saída de uma RNA em um problema de classificação binária é um valor real entre 0 e 1 para cada caso analisado. Assim, dada a saída, $0 < \hat{y} < 1$, devemos decidir em qual classe esse caso pertence e para isso fazemos o seguinte:
  - Casos são previstos como sendo da classe $c = 1$, se $\hat{y} \geq$ limiar;
  - Casos são previstos como sendo da classe $c = 0$, se $\hat{y} <$ limiar;
  - O valor do limiar é um número real entre 0 e 1, o mais usual é adotar o limar igual a 0,5.
- Observe que a exatidão definida pela equação (4) representa a fração de exemplos classificada corretamente.

➢ Para escolhermos a métrica exatidão no Keras basta incluir essa informação na etapa de compilação da RNA, que no caso de um problema de classificação binária, tem-se:

```
from tensorflow.keras import optimizers
rna.compile(optimizer='SGD', loss='binary_crossentropy',
            metrics=['accuracy'])
```

2.2 Erro absoluto médio

- O erro absoluto médio representa o quão longe a solução obtida está da solução correta.
- A forma de calcular o erro absoluto médio depende do tipo de problema.
- Em problemas de ajuste de **ajuste de funções** ⇒ um elemento do conjunto de treinamento consiste de um vetor $\mathbf{x}^{(i)}$, de dimensão (n_x, 1), e a sua saída correspondente $\mathbf{y}^{(i)}$ é um vetor de números reais de dimensão (n_y, 1).
- Nesse caso o erro absoluto médio (*mae*) é definido por:

$$mae = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j=1}^{n_y}\left|y_j^{(i)} - \hat{y}_j^{(i)}\right|\right) = \frac{1}{m}\sum_{i=1}^{m}\left\|\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)})\right\|_1 \quad (5)$$

onde $\|\mathbf{v}\|_1$ representa a norma 1 do vetor $\mathbf{v}$, ou seja:

$$\|\mathbf{v}\|_2 = |v_1| + |v_2| + \ldots |v_{n_y}| \quad (6)$$

- Para escolhermos a métrica erro absoluto médio no Keras basta incluir essa informação na etapa de compilação da RNA, ou seja:

```
rna.compile(optimizer='SGD', loss='mse', metrics=['mae'])
```

  - No comando acima a função de custo `mse` representa o erro quadrático médio visto na Aula 3, que é normalmente utilizada para problemas de ajuste de funções.

- Podemos também utilizar mais de uma métrica para avaliar a nossa RNA ⇒ para calcular tanto a exatidão como e erro absoluto médio a compilação deve ser realizada da seguinte forma:

```
rna.compile(optimizer='SGD', loss='mse', metrics=['accuracy','mae'])
```

2.3 Precisão e revocação ("precision/recal")

- **Para problemas de classificação onde o número de exemplos de uma classe é desbalanceado não se deve usar a exatidão como métrica.**
- Por exemplo, um problema de identificação de pacientes com câncer.
  - Nesse exemplo queremos identificar se um paciente tem ou não câncer, a partir de dados de exames médicos.
  - Isso consiste em um problema de classificação binária, onde:
    - $y = 1$ ⇒ paciente com câncer;
    - $y = 0$ ⇒ paciente não tem câncer.

- Para resolver esse problema é usada uma RNA e o resultado final após o treinamento da RNA é, por exemplo, uma exatidão de 99%, ou seja, a RNA acertou 99% dos diagnósticos e errou 1%.

- Sabendo que somente 0,5% dos pacientes que fazem os exames têm de fato câncer ⇒ isso pode ser considerado um bom resultado?

- Obviamente que esse não é um bom resultado e obviamente a exatidão não é uma boa métrica para esse tipo de problema.

➢ Problemas onde uma das classes aparece raramente em relação a outra a exatidão não é uma boa métrica.

➢ Então, como resolver esse problema? Que métrica usar?

A solução é usar as métricas precisão e revocação.

➢ Na Tabela 1 é apresentado um quadro dos possíveis acertos e erros das previsões da RNA. Observa-se que as previsões podem ser erradas tanto na **classificação de positivo (*y* = 1)**, quanto na **classificação de negativo (*y* = 0)**.

Tabela 1. Acertos e erros das previsões de uma RNA em um problema de classificação binária.

		Classe real	
		1	0
Classe prevista	1	Positivo Verdadeiro (*PV*)	Positivo Falso (*PF*)
	0	Negativo Falso (*NF*)	Negativo Verdadeiro (*NV*)

➢ As definições de *PV*, *NV*, *NF* e *NP* são as seguintes:

- Previsto 1, real 1 ⇒ Positivo Verdadeiro (*PV*);
- Previsto 0, real 0 ⇒ Negativo Verdadeiro (*NV*);
- Previsto 0, real 1 ⇒ Negativo Falso (*NF*);
- Previsto 1, real 0 ⇒ Positivo Falso (*PF*).

➢ A partir dos erros e acertos mostrados na Tabela 1 é fácil definir o que é precisão e revocação.

➢ Note que as previsões corretas são somente as *PV* e *NV*.

2.4 Precisão

- No caso do exemplo de pacientes com câncer, precisão é a fração de pacientes que foram detectados como tendo câncer e que de fato tem câncer.

- A precisão é calculada de acordo com a seguinte equação:

$$\text{Precisão} = \frac{PV}{\text{número previstos como positivos}} = \frac{PV}{PV + PF} \leq 1 \qquad (7)$$

2.5 Revocação

- No caso do exemplo de pacientes com câncer, revocação é a fração dos pacientes previstos como tendo câncer em relação a todos os pacientes que de fato tem câncer.

- A revocação é calculada a partir da seguinte equação:

$$\text{Revocação} = \frac{PV}{\text{número real de pacientes que de fato tem cancer}} = \frac{PV}{PV + NF} \leq 1 \qquad (8)$$

- Podemos generalizar os conceitos de precisão e revocação para qualquer problema de classificação binária.

  Considerando que a classe rara que queremos detectar tem saída real $y = 1$, então:

> **Precisão** é a relação entre a fração de previstos como sendo $y = 1$, que de fato pertencem à classe $y = 1$, em relação ao número total de previstos com pertencendo à classe $y = 1$.
>
> **Revocação** é a relação entre o número total de previstos como pertencendo à classe $y = 1$, em relação ao número total de elementos que realmente pertencem a essa classe.

- O desejado em um problema de classificação binária é ter ambos precisão e revocação altas e iguais a 1 $\Rightarrow$ mas isso nem sempre é possível. Porque?

  Porque existe um compromisso entre precisão e revocação.

- Como visto, a saída de uma RNA em um problema de classificação binária é um valor real entre 0 e 1 para cada caso analisado, que representa a probabilidade do caso pertencer à uma das classes.

- Também como já vimos, dada a saída da RNA, $0 < \hat{y} < 1$, devemos decidir em qual classe esse caso pertence, ou seja:

  - Casos são previstos como sendo da classe $y = 1$, se $\hat{y} \geq$ limiar;
  - Casos são previstos como sendo da classe $y = 0$, se $\hat{y} <$ limiar.

- **Dependendo do valor do limar utilizado teremos resultados diferentes para a precisão e a revocação ⇒ existe um compromisso entre precisão e revocação que depende do que queremos e em função disso podemos escolher o valor do limiar.**

- No exemplo de diagnóstico de câncer, se quisermos uma previsão com precisão alta, então, uma forma de obter isso é aumentar o limiar, por exemplo, para 0,7, ou seja:
  - Casos são previstos como sendo da classe $y = 1$, se $\hat{y} \geq 0{,}7$;
  - Casos são previstos como sendo da classe $y = 0$, se $\hat{y} < 0{,}7$;
  - Dessa forma, somente prevemos que um paciente tem câncer se a probabilidade for maior do que 70%;
  - Fazendo isso teremos uma precisão maior, mas também teremos uma revocação menor.

- Por outro lado, ainda no exemplo de diagnóstico de câncer, se quisermos previsões mais seguras então podemos diminuir o limiar, ou seja:
  - Casos são previstos como sendo da classe $y = 1$, se $\hat{y} \geq 0{,}3$;
  - Casos são previstos como sendo da classe $y = 0$, se $\hat{y} < 0{,}3$;
  - Dessa forma, prevemos que um paciente tem câncer para todos os casos onde a probabilidade for maior do que 30%, ou seja, vamos errar poucos diagnósticos de câncer reais;
  - Com isso teremos maior segurança na previsão, ou seja, teremos uma revocação alta, mas uma precisão baixa.

- **Concluindo:**
  - Quanto maior o limiar, maior a precisão e menor a revocação;
  - Quanto menor o limiar, maior a revocação e menor a precisão.

- Para escolhermos as métrica PV, PF, NV e NF no Keras do TensorFlow usamos na compilação o seguinte comando:

```
rna.compile('sgd', loss='mse',
            metrics=[tf.keras.metrics.TruePositives(),
            tf.keras.metrics.TrueNegatives(),
            tf.keras.metrics.FalsePositives(),
            tf.keras.metrics.FalseNegatives()])
```

- Para escolhermos as métricas precisão e revocação no Keras do TensorFlow usamos na compilação o seguinte comando:

```
rna.compile('sgd', loss='mse', metrics=[tf.keras.metrics.Recall(),
           tf.keras.metrics.Precision()])
```

2.6 Pontuação F1 ("F1 score")

- Uma métrica melhor, que combina a precisão com a revocação é a pontuação F1.

- A pontuação $F1$ transforma a precisão e a revocação em um único valor $\Rightarrow$ o que é muito melhor e mais conveniente.

- A pontuação $F1$ é definida por:

$$F1 = \frac{2PR}{P + R} \tag{9}$$

onde: P = precisão; e R = revocação.

- Observa-se que para a pontuação $F1$ ser alta, tanto a precisão quanto a revocação devem ser altas.
  - Note que $F1$ =1 somente se P e R forem ambos iguais a 1.
  - Note que se P ou R for igual a zero, então, $F1$ é igual a 0.
  - Pontuação $F1$ é uma forma de comparar precisão e revocação.

- Pontuação $F1$ é a melhor métrica para problemas de classificação onde o número de exemplos de uma classe **é desbalanceado.**

- O Keras do TensorFlow não possui a métrica pontuação F1, mas ela pode ser facilmente calculada de acordo com a equação (9) tendo a precisão e a revocação.

- **Observações:**
  - Existem inúmeras métricas disponíveis para serem usadas no Keras do TensorFlow. A lista das métricas existentes pode ser consultada no manual do TensorFlow no link https://www.tensorflow.org/api_docs/python/tf/keras/metrics/.
  - Existem também inúmeras funções de custo disponíveis para serem usadas no Keras do TensorFlow. A lista das funções de custo existentes pode ser consultada no link https://www.tensorflow.org/api_docs/python/tf/keras/losses.
  - Outras métricas e funções de custo, inclusive a métrica pontuação F1, estão também disponíveis no módulo Addons do TensorFlow. Esse módulo precisa ser instalado antes de ser usado. Mais informação sobre esse módulo pode ser obtida no link https://medium.com/tensorflow/introducing-tensorflow-addons-6131a50a3dcf.

2.7 Curva ROC

- Uma curva ROC (Curva característica de Operação do Receptor) é um gráfico que mostra o desempenho de um modelo de classificação binária para diferentes limiares. Esta curva usa dois parâmetros:
  - Taxa de positivo verdadeiro (*PV*);

- Taxa de positivo falso (*PF*).

➢ A curva ROC mostra a taxa de *PV* em função da taxa de *PF* para diferentes limiares de classificação.

➢ Reduzir o limite de classificação classifica mais itens como positivos, aumentando assim tanto os falsos positivos quanto os verdadeiros positivos. A Figura 3 mostra uma curva ROC típica.

➢ Para calcular os pontos de uma curva ROC, basta avaliar o desempenho do modelo de classificação binária para diversos limiares de classificação.

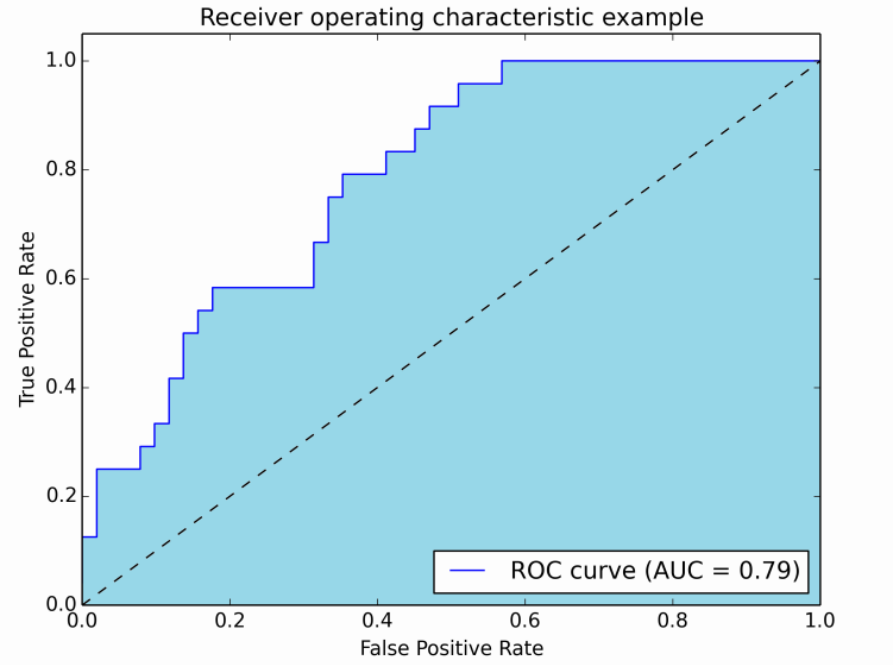

Figura 3. Exemplo de curva ROC típica (https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it).

➢ Para facilitar a análise de um modelo usando a curva ROC é usada a métrica AUC, que significa Área sob a curva ROC, ou seja, a AUC mede toda a área embaixo da curva ROC.

➢ A AUC fornece uma medida de desempenho para todos os limiares de classificação possíveis.

➢ A AUC fornece a probabilidade do modelo classificar um exemplo positivo mais alto do que um exemplo negativo.

➢ AUC varia em entre 0 e 1. Um modelo cujas previsões estão 100% erradas tem AUC igual a 0; enquanto um cujas previsões são 100% corretas tem AUC igual a 1 ⇒ ou seja, quanto maior a AUC melhor o modelo.

➢ AUC é desejável pelos dois motivos a seguir:

- AUC é invariante de escala, ou seja, ela mede quão bem as previsões são classificadas, em vez de seus valores absolutos.

- AUC é invariante do limiar de classificação, ou seja, ela mede a qualidade das previsões do modelo, independentemente do limite de classificação escolhido.

➢ No entanto, a AUC tem limitações que pode limitar a sua utilidade em certos casos, tais como:

- A invariância de escala nem sempre é desejável. Por exemplo, quando é necessário ter as probabilidades das classes, a AUC não fornece esse dado.
- A invariância do limiar de classificação nem sempre é desejável. Nos casos em que há grandes disparidades no custo de falsos negativos versus falsos positivos, pode ser fundamental minimizar um tipo de erro de classificação. Por exemplo, ao fazer a detecção de spam de e-mail, em geral deseja-se priorizar a minimização de falsos positivos (mesmo que isso resulte em um aumento significativo de falsos negativos). AUC não é uma métrica útil para este tipo de otimização.

3. Conjunto de dados com exemplos desbalanceados

➢ Para treinar uma RNA para resolver problemas de classificação com número de exemplos das classes desbalanceado deve-se modificar a função de custo para considerar o desbalanceamento no número de exemplos de cada classe.

➢ A modificação na função de custo é realizada multiplicando cada termo da função de custo por um peso inversamente proporcional ao número de exemplos de cada classe.

➢ Note que o número de termos da função de custo de um problema de classificação é igual ao número de classes que se deseja identificar. Por exemplo, a função logística (vista na Aula 4 – Classificação binária) usada para classificação binária possui dois termos, cada um referente à uma classe, conforme mostrado na equação (10).

$$J(\mathbf{W},\mathbf{B}) = \underbrace{-\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log\hat{y}^{(i)}}_{Termo\ referente\ à\ classe\ y=1} \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(1-y^{(i)}\right) log\left(1-\hat{y}^{(i)}\right)}_{Termo\ referente\ à\ classe\ y=0} \quad (10)$$

➢ A função logística modificada para um problema de classificação binária desbalanceada é dada pela equação (11).

$$J(\mathbf{W},\mathbf{B}) = \underbrace{-\frac{p_1}{m}\sum_{i=1}^{m} y^{(i)} log\hat{y}^{(i)}}_{Termo\ referente\ à\ classe\ y=1} \underbrace{-\frac{p_0}{m}\sum_{i=1}^{m}\left(1-y^{(i)}\right) log\left(1-\hat{y}^{(i)}\right)}_{Termo\ referente\ à\ classe\ y=0} \quad (11)$$

onde p_1 e p_0 são os pesos das classes $y = 1$ e $y = 0$ respectivamente.

➢ Os pesos p_0 e p_1 são inversamente proporcionais aos números de exemplos de cada classe. Por exemplo, se a classe $y = 0$ possuir 8.000 exemplos e a classe $y = 1$ possuir 2.000 exemplos, então, deve-se ter pesos $p_0 = 0{,}625$ e $p_1 = 2{,}5$, em razão da classe 0 possuir 4 vezes mais exemplos do que a classe 1.

- Esses pesos são calculados de acordo com a seguinte expressão:

$$p_i = \frac{m}{N_C n_i} \tag{12}$$

onde p_i é o peso da i-ésima classe, m é o número total de exemplos de treinamento, N_C é o número de classe e n_i é o número de exemplos da i-ésima classe.

- Os pesos da classe podem ser calculados pela função `class_weight.compute_class_weight` da bliblioteca scikit-lear (https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html). No código a seguir é mostrado como fazer isso.

```
# Importa função class_weight da biblioteca sklearn
from sklearn.utils import class_weight

# Calcula pesos das classes
pesos_classes = class_weight.compute_class_weight('balanced',
    np.unique(y), y)
```

- O método `fit` do Keras possui um parâmetro (`class_weigth`) que permite incluir na função de custo o peso de cada classe. Note que `class_weigth` é uma estrutura tipo dicionário e possui um elemento para cada classe do problema de classificação.

- Para um problema de classificação binária onde tem-se os pesos de cada classe, p_0 = 0,625 e p_1 = 2,5, o parâmetro `class_weigth` é definido de acordo com o código mostrado a seguir.

```
pesos_classes = {0:0.625, 1:2.5}
```

- Note que no caso de um problema com mais de duas classes, basta incluir mais pesos no dicionário `class_weigth`.

- Para considerar o desbalanceamento entre o número de exemplos das classes no treinamento da RNA basta incluir o parâmetro `class_weigth` no método `fit`, conforme mostrado no código a seguir.

```
history = rna.fit(x_train, y_train, batch_size=64, epochs=100,
    class_weight=pesos_classes,
    validation_data=(x_val, y_val),
    shuffle=True)
```

  - Observe que nesse comando de treinamento foi incluída a opção de embaralhar os exemplos de treinamento para tirá-los da sequência como foram definidos.