Resolvendo CAPTCHAs

Julio Trecenti

19 de julho de 2018

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

CAPTCHAs (*Completely Automated Public Turing test to tell Computers and Humans Apart*) são desafios criados com soluções fáceis de obter por humanos, mas difíceis de obter por robôs. Os CAPTCHAs nasceram entre 2000 e 2002 nos laboratórios da universidade de Carnegie Mellon (von Ahn et al., 2002), como uma tecnologia utilizada para evitar ataques *spam*.

Ao longo dos anos, os CAPTCHAs tornaram-se comuns em diversas páginas da internet, podendo ser encontrados a partir de desafios de visão, audição, operações matemáticas, entre outros. Essa popularidade também trouxe iniciativas para resolver CAPTCHAs automaticamente, a partir de inputs humanos, heurísticas codificadas manualmente e até sistemas mais sofisticados de inteligência artificial. A disputa entre geradores e resolvedores gera debates profundos até os dias de hoje.

Apesar da origem na criptografia e motivação prática, pode-se argumentar que CAPTCHAs representam um problema geral da inteligência artificial (George et al., 2017).

Essencialmente, o CAPTCHA (baseado em imagens) é uma resposta natural para pergunta de pesquisa: que aspecto da visão humana não pode ser automatizado? Essa pergunta serve não só para criar novos CAPTCHAs mas também para orientar a pesquisa em visão computacional.

Uma forma ad-hoc de responder à essa pergunta é criando desafios complexos. Por exemplo, elaborando tarefas de identificar textos dentro de imagens completamente distorcidas e cheias de ruído. No entanto, hoje somos perfeitamente capazes de resolver esse problema com altíssima taxa de acerto, muitas vezes mais altas que de humanos. Para isso, só precisamos de uma base de dados de treino suficientemente grande e modelos de aprendizado estatístico flexíveis, que chamamos de força-bruta. Discutiremos sobre isso no Capítulo 3.

Talvez a resposta não esteja na dificuldade, mas no contexto. Ao mudar o ambiente que

circunda a tarefa, mudamos a base de dados necessária para lidar com o problema. Como o levantamento de novas bases de dados de treino é custosa, essa estratégia torna a resolução por métodos de força-bruta inexequíveis. Esse racional foi a base da criação da versão dois do **reCaptcha** da Google, amplamente utilizada nos dias de hoje. O reCaptcha pode ser considerado o problema a ser batido nessa área.

E de que forma podemos avançar na pesquisa? Nossa hipótese, que é o objeto principal dessa tese, é a de que é possível desenvolver técnicas para aumentar a eficiência e a capacidade de generalização dos modelos para resolver CAPTCHAs baseados em textos, aproveitando ao máximo o que existe de informação disponível no contexto de CAPTCHAs e visão computacional. Qualquer passo positivo nesse sentido é um novo avanço na pesquisa de resolução de CAPTCHAs e, consequentemente, em problemas mais profundos de inteligência artificial.

A jornada a deste projeto de doutorado parte do problema geral dos CAPTCHAs e se encerra na apresentação e avaliação de diversas abordagens para aumentar a eficiência e a capacidade de generalização dos modelos. No caminho, faremos uma ponte entre modelos estatísticos usuais e modelos de redes neurais convolucionais, com o intuito de mostrar que essencialmente estamos apenas estendendo modelos, não abandonando a estatística clássica.

Nesse primeiro produto de qualificação, definimos o problema geral dos CAPTCHAs e suas variantes. Em seguida, mostramos a solução clássica e a solução força-bruta para o problema, apresentando os resultados obtidos até o momento. Encerramos o trabalho apresentando informalmente com os conceitos de eficiência e generalização, apontando os próximos passos que a pesquisa deve seguir.

1.1 Objetivos

O objetivo geral dessa tese é buscar formas eficientes e gerais de resolver CAPTCHAs de imagens baseados em textos.

Os objetivos específicos são:

- Definir o problema do CAPTCHA e duas variações
- Mostrar uma solução força-bruta que funciona com alta taxa de acurácia
- Realizar uma ponte teórica entre regressão logística e redes neurais convolucionais
- Desenvolver uma ferramenta para resolver CAPTCHAs que seja útil e simples de utilizar.
- Discutir e testar diversas abordagens para aumentar eficiência e capacidade de generalização dos modelos.

1.2 Resultados esperados

O projeto de doutorado já possui novos avanços relevantes nesses pontos:

- Implementação colaborativa do pacote `decryptr` na linguagem R para resolver CAPTCHAs
- Ponte teórica entre modelo de regressão logística e redes neurais convolucionais utilizando notação comum para estatísticos.
- Levantamento de lista curada de artigos sobre o tema.
- Resolução de diversos CAPTCHAs que nunca foram trabalhados no passado.

Ao final do projeto, teremos mais esses novos avanços:

- Implementação das metodologias mais recentes de forma reprodutível
- Estudo e validação de abordagens para aumentar eficiência e generalização dos algoritmos
- Investigação do uso de aprendizado por reforço e aproveitamento do oráculo

1.3 Revisão bibliográfica

A produção acadêmica sobre CAPTCHAs se agrupa em dois tópicos: geração e resolução. Os artigos de geração buscam as formas mais eficazes de criar tarefas difíceis para robôs e fáceis para humanos. Já os artigos de resolução tentam criar novos métodos para lidar com os CAPTCHAs mais comuns da internet.

(von Ahn et al., 2002) define captchas pela primeira vez. Von Ahn et al. (2003) deduz o problema de resolver CAPTCHA a partir de uma definição matemática genérica do problema da inteligência artificial. Mori and Malik (2003) apresentam uma solução rudimentar para o problema a partir de cortes em imagens. O modelo final acerta somente 33% dos casos. Yan and El Ahmad (2008a) criam um modelo de quebrar CAPTCHAs baseado em diversas heurísticas. Yan and El Ahmad (2008b) apresentam os principais problemas de utilização no desenvolvimento de CAPTCHAs concluindo, principalmente, que CAPTCHAs baseados em textos podem sim ser desafios complexos. Golle (2008) utilizam máquinas de vetor de suporte pela primeira vez. Motoyama et al. (2010) estudam o aspecto econômico dos reCaptchas, avaliando o impacto de sua utilização ampla em toda a internet. Bursztein et al. (2011) testam diversos modelos em 15 captchas diferentes e concluem que a melhor forma de criar CAPTCHAs complicados é colando as letras. Bursztein et al. (2014) mostram pela primeira vez um modelo um modelo de aprendizado por reforço, utilizando feedbacks humanos. George et al. (2017) desenvolvem um novo modelo capaz de quebrar CAPTCHAs a partir de pouquíssimos exemplos.

Capítulo 2

Problema

O problema do Captcha pode ser entendido como um problema de classificação de imagens. Especificamente, nosso interesse é criar uma função g que recebe uma imagem $\mathbf{X} = \{x_{nmr} \in [0,1]\}_{N \times M \times R}$ e retorna um vetor de índices $\mathbf{y}$, sendo que cada índice y_j corresponde a um caractere c_j, $j = 1, \ldots, L$, onde L é o número de caracteres contidos na imagem. Chamaremos L de *comprimento* do Captcha.

Podemos detalhar as afirmações anteriores em três itens, listados abaixo.

1. Nossa variável **explicativa**, a imagem, é uma matriz $\mathbf{X} = \{x_{ijk}\}_{N \times M \times R}$, em que N é o número de linhas, M é o número de colunas e R é o número de *cores*, ou *canais*.

O elemento $x_{nm\cdot}$ é denominado *pixel*. Um pixel representa a menor unidade possível da imagem. Em uma imagem colorida, por exemplo, temos $R = 3$. Nesse caso, um pixel é um vetor de três dimensões com valores entre zero e um, representando a intensidade de vermelho, verde e azul da coordenada n, m da imagem. Numa imagem em escala de cinza, temos $R = 1$ e o pixel, de uma dimensão, representa a intensidade do cinza (com 1=branco e 0=preto).

2. O objeto $C \in \mathcal{A}^L$ é um vetor de itens de um alfabeto $\mathcal{A}$ com tamanho $|\mathcal{A}|$, finito e conhecido. Esse alfabeto contém todos os possíveis caracteres que podem aparecer na imagem.
3. Nossa **resposta** $\mathbf{y} \in \{1, \ldots, |\mathcal{A}|\}^L$ é um vetor de índices de tamanho fixo. Cada elemento de $\mathbf{y}$ representa um valor do alfabeto $\mathcal{A}$.

A construção de uma função g capaz de mapear $\mathbf{y}$ a partir de uma nova imagem $\mathbf{X}$ depende de uma amostra de imagens $\mathbf{X}_1, \ldots, \mathbf{X}_S$ corretamente classificadas por $\mathbf{y}_1, \ldots, \mathbf{y}_S$. A tarefa é, portanto, obter uma estimativa $\hat{g}$ para a função g que minimiza

$$L(\hat{g}(\mathbf{X}), \mathbf{y}) = \mathbb{I}(g(\mathbf{X}) \neq \mathbf{y})$$

em que $\mathbb{I}$ é a função indicadora.

2.1 Variantes

2.1.1 Áudio

Captchas também ser encontrados na forma de áudio. Nesses casos, o usuário é condicionado a ouvir um áudio e transcrever seu conteúdo em um texto.

Com base nos Captchas analisados durante o desenvolvimento do trabalho, verificamos que Captchas de áudio são menos complexos. Por exemplo, alguns destes são formados por sons sem ruído. Ou seja, uma tabela de sons e classificações seria suficiente para quebrar os Captchas.

Nos casos com ruído, podemos utilizar duas técnicas para quebrar Captchas de áudio. A primeira é baseada em engenharia de características [ref], que extrai covariáveis dos áudios, para serem utilizadas posteriormente em um modelo de regressão clássico. O segundo método consiste em calcular o espectrograma do áudio e tratá-lo como um Captcha de imagem.

2.1.2 Covariáveis e número de respostas variável

Outra forma comum de manifestação de Captchas são imagens em que o número de letras varia. Esse problema é mais complexo que o anterior pois a variável resposta não é mais um vetor de tamanho L fixo.

Assumindo que o número de letras máximo que aparece numa imagem é fixo, uma solução para o problema de tamanho variável é considerar um caractere adicional _ no alfabeto, chamado *placeholder*, que representa a ausência do caractere em determinada posição. Assim, uma imagem com $L_{\max} = 5$, por exemplo, poderia ter uma classificação `cad5_`, que tem apenas quatro letras.

O problema dessa proposta é que a posição dos *placeholders* não importa na classificação do captcha. Ou seja, as soluções `cad5_`, `ca_d5` e `_cad5` são equivalentes. Isso aumenta a dificuldade na estimação dos parâmetros por gerar mais ótimos locais.

Outra alternativa para o problema é considerar que um de nossos interesses de predição é justamente o número de letras. Assim, teríamos

$$\tilde{\mathbf{y}} = \left[l \;\; \mathbf{y}^\top \right]^\top,$$

onde l é o número de letras do Captcha específico e $\mathbf{y}$ tem comprimento $L_{\max}$, preenchido com *placeholders* como anteriormente. A presença do número de letras como componente da variável resposta do problema permite trabalhar adequadamente com os *placeholders*, pois podemos descartá-los na função de perda (verossimilhança) nos casos em que o número de letras estimadas for menor que $L_{\max}$.

Usualmente, o problema de Captchas de tamanho variável é acompanhado pelo problema das covariáveis. Um tipo comum de Captcha com covariáveis são questões como "digite todas as letras da cor verde". Nesse caso, "verde" é uma covariável, pois é uma das informações que deve ser utilizada em conjunto com a imagem completa para predizr o texto do Captcha.

Inicialmente, pode parecer contra intuitivo considerar uma imagem e um texto como covariáveis para predizer o valor de $\mathbf{y}$, mas isso se dá de forma natural, principalmente na solução que proposta no Capítulo 3.

2.1.3 reCaptcha 2.0

O reCaptcha 2.0 desenvolvido pela Google é o *benchmark* no que se refere a geração de CAPTCHAs. Nesse desafio, o usuário é colocado para resolver diversos tipos de problemas de imagens como reconhecimento de placas de trânsito, fachadas de prédios e carros. O reCaptcha é utilizado não só para barrar robôs, como também para alimentar os modelos estatísticos da própria Google para seus produtos, como, por exemplo, os carros autônomos.

A dificuldade em resolver esses desafios consiste na pluralidade de contextos, não na dificuldade dos desafios (Goodfellow et al., 2013). A tarefa de identificar placas de trânsito em imagens é complicada, mas pode ser resolvida com alta taxa de acurácia utilizando-se modelos adequados. No entanto, o reCaptcha pode facilmente mudar a tarefa para identificação de fachadas, ou então identificação de felinos. Do ponto de vista estatístico, cada uma dessas tarefas exige uma base de dados diferente.

Na prática, o reCaptcha cria um jogo em que o autor do CAPTCHA tem muitas vantagens. O tempo necessário para criar uma nova tarefa é menor que o tempo necessário para resolvê-lo. Até hoje não existe uma solução geral para o reCaptcha 2.0 com taxas satisfatórias de acurácia.

A principal forma de resolver o reCaptcha 2.0 é através da exploração de falhas no sistema de intificação de robôs ou a possibilidade de receber tarefas por áudio [sivakorn2016m]. Outras abordagens utilizam modelos robustos como o ImageNet para identificação genérica de objetos nas imagens, mas sem taxas de sucesso expressivas [mach2017automaticky].

Capítulo 3

Solução

3.1 Segmentação e classificação

Um problema de resolver o CAPTCHA diretamente é que a variável resposta $\mathbf{y}$ tem um número exponencial de combinações. Na formulação do capítulo anterior, nossa resposta é uma palavra de L caracteres, sendo que cada caractere c_j pode ter $|\mathcal{A}|$ valores. Nessa construção, o total de combinações é $|\mathcal{A}|^L$.

Por exemplo, um CAPTCHA com $L = 6$ letras e $|\mathcal{A}| = 36$ possibilidades em cada letra (26 letras do alfabeto e 10 algarismos), possui um total de 2.176.782.336 (> 2 bilhões) combinações. Modelar essas imagens diretamente através de uma única variável resposta categórica é inviável.

Por isso, uma forma de resolver CAPTCHAs é separando o problema em duas tarefas: segmentar e classificar. A tarefa de segmentação consiste em receber uma imagem com várias letras e detectar pontos de corte, separando-a em várias imagens de uma letra. Já a classificação consiste em receber uma imagem com uma letra e identificar o caractere correspondente. Nesse caso, a resposta é reduzida para $|\mathcal{A}|$ categorias, que cresce linearmente e, portanto, tratável.

A literatura mostra através de estudos empíricos que a tarefa de segmentar é mais difícil do que a tarefa de classificar (Bursztein et al., 2014). Isso acontece porque o problema de classificação de letras segmentadas é similar ao problema de reconhecimento de caracteres (*Optical Character Recognition*, OCR), que é amplamente estudado e pode ser considerado resolvido. A segmentação, no entanto, é um problema em aberto e faz parte da literatura de oclusão de objetos em visão computacional.

Por esse motivo, os desenvolvedores de CAPTCHAs de imagens baseadas em texto têm explorado métodos de dificultar a segmentação. As principais formas são i) colar os caracteres e

ii) adicionar linhas ligando os dígitos. Essas técnicas são combinadas com a adição de ruído e distorção de caracteres para compor a imagem final.

Vamos usar como exemplo o CAPTCHA do Tribunal de Justiça de Minas Gerais (TJMG). Nesse caso, temos $L = 4$ e $|\mathcal{A}| = 10$, apenas os dez algarismos.

A Figura 3.1 mostra um exemplo do captcha do TJMG. Podemos notar a utilização de distorção de catacteres e adição de linhas ligando os dígitos como formas de evitar a resolução automática.

Figura 3.1: CAPTCHA do TJMG.

Nesse caso, podemos resolver o problema da segmentação realizando cortes fixos na imagem. Podemos também limitar os eixos x, tirando os espaços vazios à esquerda e à direita e y, removendo espaços superiores e inferiores. Por último, transformamos a imagem em escala de cinza. O resultado dessas operações de pré-processamento estão na Figura 3.2.

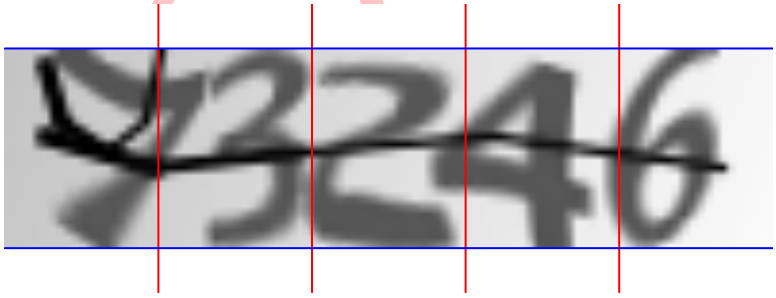


Figura 3.2: CAPTCHA do TJMG após segmentação.

O resultado são cinco imagens de dimensões $26\mathrm{x}20$, associadas a cada caractere. O próximo passo é transformar o banco de dados num formato tratável por modelos tradicionais de regressão. Para isso, colocamos cada pixel em uma coluna da nossa base de dados. No caso do TJMG,

Tabela 3.1: Base de dados montada a partir de imagem segmentada.

y	xy01_01	xy01_02	xy01_03	xy01_04	xy01_05	xy01_06
7	0.769	0.769	0.769	0.769	0.769	0.773
3	0.005	0.141	0.316	0.430	0.356	0.319
2	0.846	0.851	0.830	0.796	0.800	0.842
4	0.886	0.886	0.890	0.890	0.890	0.890
6	0.925	0.925	0.929	0.929	0.929	0.933

Tabela 3.2: Tabela de acertos e erros.

y	0	1	2	3	4	5	6	7	8	9
0	156	.	.	.	.	.	.	.	.	.
1	.	160	.	.	.	.	.	.	.	.
2	.	.	147	.	.	.	.	.	.	.
3	.	.	1	140	.	.	.	.	.	.
4	.	2	.	.	150	.	.	.	.	.
5	.	.	.	.	.	153	.	.	.	.
6	.	.	.	.	.	.	143	.	.	.
7	.	.	.	.	.	.	.	152	.	.
8	.	.	.	2	1	.	.	.	139	.
9	.	.	.	.	.	.	.	.	.	154

cada CAPTCHA gera uma tabela de 5 linhas e 520 ($26 * 20$) colunas. A Tabela 3.1 mostra as primeiras seis colunas dessa base.

Agora basta rodar o mesmo para toda a base de treino e rodar um modelo. Nesse exemplo, utilizamos uma base de 1500 CAPTCHAs classificados. O resultado após o pré-processamento é uma base com 7500 linhas e 520 colunas. Escolhemos manter 6000 linhas para treino e as 1500 restantes para teste. Utilizamos um modelo de florestas aleatórias para o exemplo (Breiman, 2001).

O resultado do modelo pode ser verificado na Tabela 3.2, que mostra os observados *versus* preditos na base de teste. O acerto foi de 99.6% em cada caractere. Assumindo que o erro não depende da posição do caractere no CAPTCHA, o acerto para a imagem completa é de aproximadamente 98%.

O resultado para o TJMG é bastante satisfatório, mas não generaliza para outros CAPTCHAs. Tome por exemplo o CAPTCHA da Receita Federal (RFB) da Figura 3.3. Nesse caso, a posição

dos caracteres muda significativamente de imagem para imagem, e assim fica difícil cortar em pedaços.

Figura 3.3: CAPTCHA Receita Federal

A mesma técnica aplicada ao CAPTCHA RFB apresentou acerto de 78.8% do caractere, o que equivale a apenas 23.8% de acerto para toda a imagem. Claro que seria possível melhorar o poder preditivo com ajustes nos hipeparâmetros do modelo, mas o problema essencial nesse caso está na qualidade segmentação, e não na classificação dos caracteres.

Outro problema dessa técnica é que ela é incapaz de trabalhar com CAPTCHAs de comprimento variável. Nesse caso, seria necessário construir um modelo não supervisionado para identificar a posição das letras, o que adiciona um grau a mais de complexidade na resolução do CAPTCHA.

Por isso, faz-se necessária uma abordagem que trabalha com problema completo, sem passar explicitamente pela fase de segmentação. Ao invés de cortar a imagem, vamos extrair detalhes da imagem completa automaticamente e utilizar essas características como variáveis preditoras num modelo de regressão. Chamaremos essa abordagem de *força bruta*.

3.2 Força-bruta

A abordagem de força bruta utiliza redes neurais convolucionais. Para explicar o funcionamento dessa técnica, vamos primeiro apresentar definições para redes neurais e para a operação de convolução. Em seguida, vamos juntar os dois conceitos para construir o modelo utilizado nos CAPTCHAs.

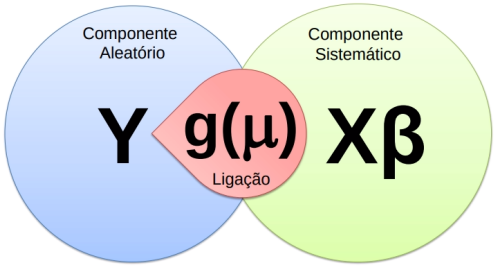
3.2.1 Redes neurais

Uma rede neural pode ser entendida como uma extensão de modelos lineares generalizados com a adição de uma arquitetura aos componentes do modelo. Para mostrar esse conceito, vamos partir da definição de um modelo regressão logística até construir uma rede neural com camadas ocultas.

3.2.1.1 Regressão logística

O modelo linear generalizado é composto por três elementos: componente aleatório, componente sistemático e função de ligação.

O componente aleatório é uma variável aleatória com distribuição pertencente à família exponencial, que dá origem à verossimilhança do modelo. O componente sistemático é uma combinação linear das variáveis preditoras com um vetor de parâmetros. A função de ligação é uma operação que leva a componente sistemática no valor esperado da componente aleatória. Uma forma comum de definir a ligação é propondo uma função com domínio nos números reais e contradomínio igual ao suporte do componente aleatório. Dessa forma, não é necessário impor restrições aos parâmetros da componente sistemática para que os valores ajustados variem na mesma faixa que o componente aletório.



No exemplo da regressão logística, o componente aleatório tem distribuição Bernoulli com média μ. O componente sistemático é a combinação linear $\mathbf{X}\boldsymbol{\beta}$ e a função de ligação é a inversa de

$$g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$$

A partir de uma amostra $y_1, \ldots, y_n$ e observando que $\mu_i = g^{-1}(\mathbf{X}_i\boldsymbol{\beta})$, a verossimilhança do modelo é dada por

$$\mathcal{L}(\boldsymbol{\beta}|\mathbf{y}) = \prod_{i=1}^{n} f(y_i|\boldsymbol{\beta}) = \prod_{i=1}^{n} \mu_i^{y_i}(1-\mu_i)^{1-y_i}$$

A log-verossimilhança é dada por

$$l(\boldsymbol{\beta}|\mathbf{y}) = \sum_{i=1}^{n} y_i \log(\mu_i) + (1-y_i)\log(1-\mu_i)$$

Uma forma útil de olhar para a verossimilhança é a partir da *função desvio*, dada por

$$D(\mathbf{y}|\boldsymbol{\beta}) = l(\mathbf{y}|\mathbf{y}) - l(\boldsymbol{\beta}|\mathbf{y}),$$

onde $l(\mathbf{y}|\mathbf{y})$ é a verossimilhança do modelo saturado, ou seja, calculada com $\mathbf{y}$ no lugar de $\boldsymbol{\mu}$. A partir de um modelo ajustado, a função desvio pode ser interpretada como a distância entre a verossimilhança do modelo ajustado e a verossimilhança do modelo com um parâmetro para cada observação.

Uma propriedade interessante da função desvio é que ela equivale à divergência de Kullback-Leibler. Por exemplo, para duas variáveis aleatórias com distribuição Bernoulli de parâmetros p e q, respectivamente, a divergência de Kullback-Leibler é dada por

$$D_{KL}(p||q) = p \log\left(\frac{p}{q}\right) + (1-p)\log\left(\frac{1-p}{1-q}\right)$$

É fácil ver que

$$\begin{aligned} D(\mathbf{y}|\boldsymbol{\beta}) &= \sum_{i=1}^{n} y_i \log(y_i) + (1-y_i)\log(1-y_i) - \sum_{i=1}^{n} y_i \log(\mu_i) + (1-y_i)\log(1-\mu_i) \\ &= \sum_{i=1}^{n} y_i \log\left(\frac{y_i}{\mu_i}\right) + (1-y_i)\log\left(\frac{1-y_i}{1-\mu_i}\right) \\ &= \sum_{i=1}^{n} D_{KL}(y_i||\mu_i) \\ &= D_{KL}(\mathbf{y}||\boldsymbol{\mu}). \end{aligned}$$

Outra propriedade interessante é que o desvio identifica unicamente a verossimilhança do modelo. De fato, podemos reformular a definição do modelo linear generalizado a partir da especificação do desvio ou da divergência de Kullback-Leibler no lugar do componente aleatório. Essa propriedade será útil na comparação com redes neurais.

Os estimadores de máxima verossimilhança de $\boldsymbol{\beta}$ são os mesmos que minimizam a função desvio. Graças à concavidade da divergência de Kullback-Leibler, Isso pode ser feito igualando os componentes do gradiente do desvio a zero e isolando os valores de $\boldsymbol{\beta}$:

$$\nabla_{\beta} D(\mathbf{y}|\boldsymbol{\beta}) = \mathbf{0}$$

Como não é possível realizar essa operação analiticamente, utilizamos métodos iterativos. Existem dois principais métodos iterativos concorrentes: a descida de gradiente e o método de Newton-Raphson. No paradigma de modelos lineares generalizados, o método de Newton-Raphson é mais comum pois i) ele utiliza a segunda derivada e converge mais rápido que o método da descida de gradiente, que utiliza somente a primeira e ii) é possível demonstrar que ele equivale à aplicação iterada de *mínimos quadrados ponderados*, o que facilita significativamente a implementação da solução. No paradigma de redes neurais, a descida de gradiente é mais comum por conta das vantagens *backpropagation*, que veremos na próxima subseção.

Em resumo, podemos concluir que

1. Um modelo linear generalizado pode ser definido por três componentes: a divergência de Kullback-Leibler, o preditor linear e a função de ligação.
2. A estimação dos parâmetros do modelo é realizada via descida de gradiente ou Newton-Raphson.

Em seguida, veremos que a rede neural aparece quando utilizamos o componente sistemático e a função de ligação várias vezes.

3.2.1.2 Extensão para redes neurais

Uma forma de estender o modelo linear generalizado é considerando que o resultado da função de ligação aplicada ao componente sistemático é uma nova covariável z. Assim, temos

$$\begin{aligned}\mathbf{z} &= g^{-1}(\mathbf{X}\boldsymbol{\beta}) \\ \boldsymbol{\mu} &= g^{-1}(\alpha_2\mathbf{1} + \beta_2\mathbf{z}) = g^{-1}([\mathbf{1}\ \mathbf{z}]\boldsymbol{\beta}_2),\end{aligned}$$

em que $\boldsymbol{\beta}_2 = [\alpha_2\ \ \beta_2]^\top$. Agora, podemos aumentar o número de covariáveis $\mathbf{z}$ para k covariáveis, de modo que

$$\begin{aligned}\mathbf{z}_j &= g^{-1}(\mathbf{X}\boldsymbol{\beta}_1^j) \\ \boldsymbol{\mu} &= g^{-1}(\mathbf{Z}\boldsymbol{\beta}_2),\end{aligned}$$

onde $\mathbf{Z} = [\mathbf{1}\ \mathbf{z}_1\ \ldots\ \mathbf{z}_k]$. O modelo espeficiado dessa forma também é chamado de *multilayer perceptron*, ou MLP.

Mesmo com essa mudança, função desvio permanece a mesma, já que construída a partir de $\boldsymbol{\mu}$. A única diferença é que agora ela é uma função de $\boldsymbol{\beta}_1^j$, $j = 1, \ldots, k$ e β_2. O ajuste do modelo é realizado da mesma forma:

$$\nabla_{\{\boldsymbol{\beta}_1^1,\ldots,\boldsymbol{\beta}_1^k,\boldsymbol{\beta}_2\}} D(\mathbf{y}|\boldsymbol{\beta}_1^1,\ldots,\boldsymbol{\beta}_1^k,\boldsymbol{\beta}_2) = \mathbf{0}$$

A vantagem dessa extensão é que adicionamos não linearidade ao modelo. Isso nos permite modelar relações mais complexas entre as preditoras e a resposta, o que pode resultar em melhores predições. De fato, é possível demonstrar que uma rede neural com uma camada oculta pode estima qualquer função contínua entre $\mathbf{X}$ e $\mathbf{y}$. A desvantagem é que a estimação via Newton-Raphson é complicada de calcular.

É nesse momento que aparecem as vantagens da descida de gradiente. Primeiro, defina $\boldsymbol{\beta} = \{\boldsymbol{\beta}_1^1,\ldots,\boldsymbol{\beta}_1^k,\boldsymbol{\beta}_2\}$. Utilizando a regra da cadeia, a derivada parcial da função desvio em relação a $\beta_{1,l}^j$ é dado por

$$\frac{\partial D(\mathbf{y}|\boldsymbol{\beta})}{\partial \beta_{1,l}^j} = \sum_{i=1}^{n} \frac{\partial D(\mathbf{y}|\boldsymbol{\beta})}{\partial z_{j,i}} \frac{\partial z_{j,i}}{\partial \beta_{1,l}^j}.$$

As derivadas em relação aos elementos de $\boldsymbol{\beta}_2$ ocorrem diretamente, como na especificação em apenas um nível. Todas essas derivadas são fáceis de calcular e têm forma analítica definida. A aplicação da regra da cadeia de forma iterada nesse contexto é chamada de *backpropagation*.

3.2.1.3 Sinônimos e generalizações

A literatura de redes neurais costuma trocar o nome função de ligação por *ativação*. Isso ocorre por motivos históricos, já que as redes neurais foram inicialmente inspiradas na ativação de sinapses de neurônios. No contexto de redes neurais, o objetivo da função de ativação não é, necessariamente, modificar a faixa de variação do contradomínio, pois o resultado após a função pode ser uma nova covariável. Isso sugere a existência de certa liberdade na escolha de ativações. A única restrição é que a função de ativação deve ser não linear, pois, se fosse linear, a aplicação de várias camadas de funções resultaria numa única combinação linear. As ativações mais populares são aquelas que têm derivadas simples.

Já a verossimilhança ou o desvio são substituídos por uma *função de perda*. A natureza probabilística do modelo é considerada indiretamente através da função desvio, como vimos anteriormente. No entanto, ao invés de trabalhar com o desvio, os pesquisadores de redes neurais definem genericamente uma função de perda que mensura uma discrepância entre os valores observados e estimados. Uma escolha razoável de função de perda é a própria divergência de Kullback-Leibler, calculada com base no suporte da variável resposta, gerando a função desvio. No entanto, dependendo da aplicação, podemos escolher outras perdas, que podem gerar distribuições de probabilidades sem formato analítico específico.

Por último, a aplicação de camadas de não-linearidades podem ser representadas através de um grafo direcionado acíclico. Essa representação é vantajosa por dois motivos. O primeiro é que a aplicação facilita a especificação e entendimento do modelo e seus parâmetros, que podem ficar com notação carregada na especificação por fórmulas matemáticas. A segunda é que é possível utilizar conhecimentos de teoria dos grafos para aumentar a eficiência dos algoritmos. Por exemplo, é possível aproveitar parte dos cálculos do *backpropagation* na obtenção das derivadas parciais da função de perda (Abadi et al., 2016).

Em resumo, podemos concluir que

1. Uma rede neural é uma extensão de modelos lineares generalizados que aplica combinações lineares e funções de ligação de forma iterada.
2. A estimação dos parâmetros é realizada por descida de gradiente, explorando as vantagens do backpropagation.
3. Funções de ligação são chamadas de funções de ativação.
4. A função desvio é substituída por funções de perda mais gerais.
5. A aplicação iterada de operações pode ser representada por um grafo direcionado acíclico.

Existem diversas formas de definir, desenhar e apresentar os conceitos básicos de redes neurais e a descida de gradiente. As melhores são apresentadas em blogs, vídeos e aplicativos, onde as operações são apresentadas de forma interativa. O racional apresentado nesse texto buscou mostrar a relação intrínseca entre a regressão logística e as redes neurais.

3.2.2 A operação de convolução

Convolução em imagens é uma operação usada nas áreas de *visão computacional* e *processamento de sinais*. Ela é utilizada para detectar padrões e aplicar filtros em imagens. Na prática, o que ela faz é calcular um novo valor para um pixel na posição (i, j) de uma imagem com base nos valores dos pixels da vizinhança.

Uma forma organizada de fazer essa soma ponderada é criando uma matriz de pesos. Com ela, não é necessário procurar os pontos da vizinhança. Para cada ponto (i, j), obtemos a matriz de vizinhança, multiplicamos pontualmente pela matriz de pesos e somamos os valores resultantes. Chamaremos essa matriz de pesos de **kernel**.

Considere

$$K = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

e a seguinte imagem:

Tome por exemplo o ponto $(i, j) = (12, 16)$. A vizinhança 3x3 em torno desse ponto é dada por

$$P_{i,j} = \begin{bmatrix} 0.98 & 0.53 & 0.79 \\ 0.97 & 0.99 & 1.00 \\ 0.98 & 1.00 & 1.00 \end{bmatrix}$$

A operação de convolução é feita da seguinte forma:

$$\begin{aligned}(P_{12,16} * K)_{12,16} = k_{1,1}p_{11,15} + k_{1,2}p_{11,16} + k_{1,3}p_{11,17} + \\ + k_{2,1}p_{12,15} + k_{2,2}p_{12,16} + k_{2,3}p_{12,17} + \\ + k_{3,1}p_{13,15} + k_{3,2}p_{13,16} + k_{3,3}p_{13,17}\end{aligned}$$

Esse é o valor a ser colocado no ponto (i, j). Isso funciona em todos os pontos que não estão na borda da imagem.

Existem duas formas de trabalhar com as bordas da imagem. A primeira é preenchendo as bordas com zeros, de forma a considerar apenas os pontos da imagem. A segunda é descartar os pontos da borda e retornar uma imagem menor, contendo somente os pixels em que foi possível aplicar todo o kernel.

No nosso caso, o resultado da convolução fica como na Figura 3.4. Essa matriz não foi escolhida por acaso. Ela serve para destacar padrões horizontais da imagem. Como a primeira linha é formada −1s e a última é formada por 1s, a matriz fica com valor alto se a parte de cima do pixel for preta e a parte de baixo for branca (`grande * 1 + pequeno * (-1)`). A parte destacada da imagem acabou sendo os olhos (pois temos maior concentração de pixels pretos ali), além das extremidades superior e inferior do rosto.

Figura 3.4: Figura após aplicação de convolução.

Aplicando o kernel vertical

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix},$$

a parte destacada do rosto são as extremidades dos lados:

A aplicação de convoluções em CAPTCHAs é direta. Nesse caso, vamos adicionar uma constante numérica ao resuldado da convolução. Isso pode auxiliar na visualização, pois controlamos os valores que ficam dentro do intervalo $[0, 1]$. Mais adiante veremos que esse será o intercepto da regressão.

Vamos partir do CAPTCHA da RFB abaixo

Esse é o resultado de adicionar o kernel vertical e bias de `0.6`.

Em seguida observamos o kernel horizontal. Note que identificamos padrões das linhas horizontais que tentam atrapalhar a visão das letras.

Também vamos introduzir uma função chamada **ReLu**. ReLu significa *Restricted Linear Unit* e é uma função que zera tudo o que é negativo e mantém tudo aquilo que é positivo inalterado. Ou seja,

$$ReLu(x) = \frac{x + |x|}{2}$$

ReLu não é útil para visualização da imagem, pois a substituição de valores negativos por zero já é feita automaticamente. No entanto, podemos aplicar várias convoluções iteradamente e separá-las por aplicações da função ReLu. Como a função ReLu é não linear, essa iteração gera resultados que não seriam possíveis de obter somente com aplicações da operação convolução.

Na prática, o que queremos é treinar os valores do kernel aplicado, buscando obter imagens transformadas que aumentem o poder preditivo. Nesse sentido, a aplicação de convoluções, soma de constantes e ReLu são as operações que substituem a multiplicação de matrizes, adição de intercepto e aplicação da função de ligação na regressão logística, respectivamente. Ou seja, uma rede neural convolucional é apenas uma forma diferente de implementar os conceitos.

3.2.3 Redes neurais convolucionais

Considere uma observação de uma imagem com 2x2 pixels abaixo. Note que se o interesse for utilizar essa matriz numa regressão logística, teríamos uma linha de nossa base de dados, com nove colunas. Ou seja, a regressão teria nove parâmetros associados.

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$$

Considere agora o kernel W, também 3x3:

$$K = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}$$

A operação convolução resulta numa nova matriz 3x3, em que cada elemento é uma combinação linear de elementos de P e K. De fato, é possível mostrar que o resultado da convolução é o resultado de uma multiplicação de matrizes obtida através da *matriz circulante* de K (Gray, 2006). Ou seja, nesse caso, estamos fazendo uma nova regressão logística, mas com os valores dos dados modificados.

Se, ao invés disso, considerarmos a matriz 2x2,

$$K = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix}$$

estamos na prática reduzindo o problema de regressão logística para apenas quatro parâmetros.

O modelo força-bruta é uma adaptação do clássico modelo LeNet-5 (LeCun et al., 2015). Esse modelo aplica convolução 3 vezes consecutivas, adicionando o viés e a função ReLu em cada nível. Após cada convolução, também aplicamos uma operação chamada *max pooling*, que reduz a resolução da imagem, tomando o valor máximo da vizinhança de cada ponto. Após a aplicação das convoluções, as imagens são remodeladas no formato retangular padrão (uma linha por imagem) e aplicamos duas camadas de redes neurais comuns, como vimos anteriormente.

Após a realização de todas as operações, os números resultantes não estão entre zero e um. Por isso, aplicamos a ativação *softmax*, que é a generalização da ativação logística, mas para uma resposta com vários resultados possíveis

$$softmax(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

Em resumo, as operações que realizamos na rede neural convolucional são

1. Tomar o input inicial (imagem).

Tabela 3.3: Resultados da aplicação dos modelos.

Imagem	Nome	Taxa de acerto
imgs/rfb.png	RFB	98%
	TRT	98%
	TJMG	100%
	RSC	99%
	CADESP	98%

2. Multiplicar (convoluir) por matrizes de pesos W.
3. Adicionar um viés (ou intercepto) b.
4. Aplicar uma função de ligação (ou ativação), por exemplo ReLu.
5. Reduzir a resolução do resultado.
6. Voltar para 2 várias vezes.
7. Tomar os pesos finais e normalizar (usando a operação *softmax*) para obter probabilidades dos resultados.

3.2.4 Resultados

Até o momento, aplicamos os modelos de redes neurais convolucionais para cinco CAPTCHAs distintos. Os modelos foram treinados a partir de bases de treino com aproximadamente dez mil exemplos para cada CAPTCHA. Os resultados da aplicação dos modelos estão na Tabela 3.3. Essas taxas foram calculadas com base em 100 novos CAPTCHAs baixados da internet após o ajuste do modelo. Podemos observar que as taxas de acerto são todas muito próximas de 100%. No mínimo essas taxas estão muito próximas do que seres humanos conseguiriam acertar.

Os resultados positivos da aplicação dos modelos força-bruta pode motivar a pergunta: o problema está completamente resolvido? De fato, podemos dizer que CAPTCHAs de imagem baseados em textos são problemas resolvidos, desde que exista uma base de dados classificada. No entanto, esses modelos não funcionam para novos CAPTCHAs e também erram se fizermos pequenas alterações nas imagens. Veremos mais detalhes desse problema no próximo capítulo.

Capítulo 4

Eficiência e generalização

A modelagem via redes neurais convolucionais apresenta resultados satisfatórios, mas tem dois problemas: eficiência e generalização.

O problema de eficiência está relacionado com o fato de que a quantidade de imagens classificadas necessária para obter bom poder preditivo é alta. A partir de nossos testes, identificamos que são necessários em torno de dez mil imagens classificadas para obter um modelo com taxa de acerto maior que 90%.

Já o problema de generalização implica que um modelo ajustado para um tipo CAPTCHA não funciona para outro, ainda que esses tipos sejam muito similares. Na verdade, esses modelos sofrem do problema de *aprender* versus *decorar* (Zhang et al., 2016). Isso significa que pequenas modificações na imagem original, e.g. inclusão de ruído gaussiano na imagem, podem resultar em predições completamente diferentes.

Os dois problemas não são independentes. Se criarmos um modelo que generaliza, é razoável afirmar que a quantidade de dados necessária para se obter bom poder preditivo numa nova imagem se reduz.

A Tabela 4.1 mostra algumas possíveis abordagens para resolver esses problemas.

Tabela 4.1: Possíveis soluções para problemas de eficiência e generalização.

Eficiência	Generalização
Feedback	Ensemble
Reciclagem	Ruído
Enriquecimento	

A **reciclagem** consiste na aplicação de métodos de *data augmentation* usuais em processamento de imagens. Esses métodos consistem em gerar novas imagens a partir das imagens originais, mas adicionando rotações, reflexões e diferentes níveis de zoom. A vantagem da reciclagem é a possibilidade de aumentar a base de treino sem aumentar a quantidade de classificações manuais, o que em tese aumenta a eficiência do aprendizado do modelo.

A aplicação de **ruídos** na imagem segue o mesmo princípio da reciclagem, mas tem foco na generalização. Ruídos podem ser adicionados através de distorções da imagem, ruídos aleatórios e oclusões. O ruído adiciona robustez nas predições, o que em tese possibilitaria que um modelo implementado para um CAPTCHA específico possa ser utilizado para um caso similar.

A utilização de técnicas de **ensemble** visa compartilhar os parâmetros de um modelo de CAPTCHA em outro modelo. A arquitetura moderna de redes neurais possibilita o encaixe de modelos pré-treinados em novas bases de treino. Por exemplo, é possível aproveitar as camadas de redes convolucionais do captcha RFB como inputs adicionais para modelo do TRT. Esses parâmetros podem ser considerados fixos ou podem entrar na verossimilhança do modelo e ser atualizado. Em tese, isso permitiria a criação de apenas um grande modelo para resolver todos os CAPTCHAs.

O **enriquecimento** surge do fato de que origem de novos parâmetros a serem adicionados a um modelo não precisam ser de CAPTCHAs ajustados anteriormente. Esses parâmetros podem vir de ferramentas genéricas de reconhecimento de caracteres (OCR) ou mesmo de bases de dados de caracteres. George et al. (2017) realizaram essa investigação com resultados satisfatórios e conseguiram reduzir o tramanho da base de treino significativamente.

Um mecanismo de **feedback** é uma técnica usada de aumentar a eficiência do modelo ao adicionar informações ao modelo quando a predição falha. As informações são incluídas no modelo através de técnicas de aprendizado por reforço (Sutton and Barto, 1998).

Existem duas formas de implementar mecanismos de feedback: automática e manual.

A forma automática aproveita as vantagens da existência de um *oráculo* de baixo custo para aumentar o tamanho da base de dados automaticamente. O oráculo é uma função que recebe uma imagem e uma predição e informa se a predição está correta ou incorreta. O oráculo tem custo baixo de utilização pois existe em praticamente todos os sites da internet, pois geralmente o CAPTCHA está associado a um formulário de consulta (e.g. uma consulta processual), que verifica se o CAPTCHA foi corretamente resolvido. O problema do oráculo é que, quando algoritmo erra, não sabemos quais são as letras que foram preditas incorretamente. Por exemplo, num caso com $L = 6$ letras e $|\mathcal{A}| = 35$, a informação que o oráculo passa quando o algoritmo erra é que o valor correto do CAPTCHA não é uma das 36^6 possibilidades. Além disso, em muitos sites só é possível testar a predição do CAPTCHA uma vez. Isso sugere que a informação do oráculo deve ser aliada a heurísticas para que seja útil.

A forma manual utiliza a análise de humanos para inclusão de informações sobre cortes da imagem e erro das letras (Bursztein et al., 2014). Uma pergunta de pesquisa interessante nesse sentido seria: qual é a mínima intervenção suficiente para fazer o modelo aprender? Isso pode ser resolvido testando diversos inputs manuais e comparar o tempo de realização do feedback com o ganho em poder preditivo.

4.1 Próximos passos

Na tese de doutorado, vamos utilizar reciclagem, adição de ruídos, ensemble e enriquecimento para construir modelos mais eficientes e robustos. Pretendemos testar os impactos dessas técnicas na relação entre tamanho da base de treino e poder preditivo do modelo.

O grande desafio da pesquisa será a investigar a forma automatizada de feedback. Caso seja possível evitar completamente o input humano para resolver os CAPTCHAs, o problema de aprendizado de CAPTCHAs baseados em texto estará completamente resolvido. Se não, vamos estudar quais são os limites da aplicação automatizada e buscar métodos que misturem eficientemente as duas formas de feedback.

Apêndice A

Pacote decryptr

Uma da nossas iniciativas principais nesta tese é a criação do pacote decryptr. Abaixo mostramos como usar algumas das funções principais desse pacote.

Até o momento o `decryptr` possui as seguintes restrições:

1. Apenas imagens `jpg` ou `png`.
2. Uma imagem possui apenas números e letras.
3. A quantidade de caracteres de um CAPTCHA é fixa.
4. Dois CAPTCHAs de mesma origem têm sempre as mesmas dimensões.

O `decryptr` ainda não está no CRAN. Isso significa que para instalá-lo é necessário utilizar o `devtools`:

```
devtools::install_github('decryptr/decryptr')
```


A.1 Funções do `decryptr`

As funções principais do `decryptr` são

- `download_captcha()`: baixar imagens da web.
- `read_captcha()`: adicionar metadados úteis a uma string com o caminho do CAPTCHA.
- `load_captcha()`: carregar a imagem na memória.
- `plot.captcha()`: método S3 para desenhar o CAPTCHA na tela.
- `classify()`: método S3 para classificar CAPTCHAs manualmente.
- `load_model()`: carregar um modelo já ajustado e armazenado no pacote `decryptrModels`

- `train_model()`: método S3 para ajustar um modelo baseado em LeNet-5 para os CAPTCHAs.
- `decrypt()`: método S3 para classificar um CAPTCHA a partir de um modelo ajustado e um caminho de imagem.

A.1.1 Fluxo de utilização

O modo de uso planejado do `decryptr` está descrito na Figura A.1.

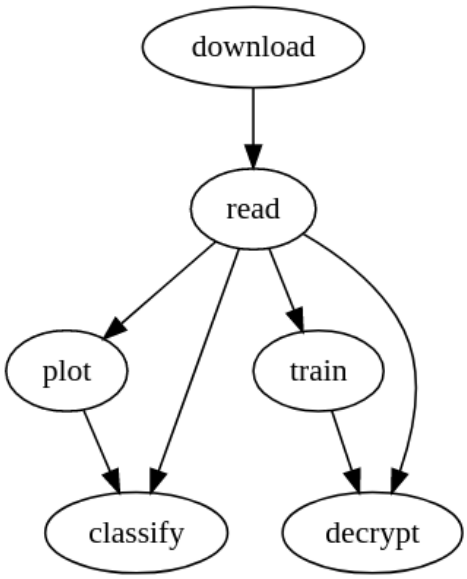


Figura A.1: Fluxo de utilização do pacote 'decryptr'.

A.1.2 Download

A função `download_captcha()` tem cinco parâmetros:

- `url=` o link do CAPTCHA que queremos baixar.
- `n=` a quantidade de CAPTCHAs a serem baixados.
- `path=` a pasta que queremos salvar a imagem.
- `secure=` se `TRUE`, fará o download com a opção `ssl_verifypeer = FALSE` (veja esse post)

- `ext=` extensão do arquivo (`jpg`/`jpeg` ou `png`).

Para facilitar a utilização do `decryptr`, adicionamos algumas atalhos do tipo `download_captcha("nome")`, que já contêm os padrões para download de alguns sites específicos:

- `download_captcha("rfb")`: Consulta de CNPJ da Receita federal.
- `download_captcha("saj")`: Sistema SAJ (vários Tribunais Estaduais).
- `download_captcha("tjmg")`: Tribunal de Justiça de Minas Gerais.
- `download_captcha("tjrj")`: Tribunal de Justiça do Rio de Janeiro.
- `download_captcha("tjrs")`: Tribunal de Justiça do Rio Grande do Sul.
- `download_captcha("trt")`: Tribunais Regionais do Trabalho.

Exemplo:

```
library(decryptr)
# salva arquivo em ./imgs/tjmg/captcha<id>.jpeg
arq <- download_captcha("tjmg", n = 1, path = 'imgs/tjmg')
```

A.1.3 Visualização

Para plotar um CAPTCHA basta ler o arquivo com `read_captcha()` e depois usar a função `plot()`. Exemplo:

```
library(decryptr)
'imgs/tjmg/captcha2f6e4f7825d6.jpeg' %>%
  read_captcha() %>%
  dplyr::first() %>%
  plot()
```

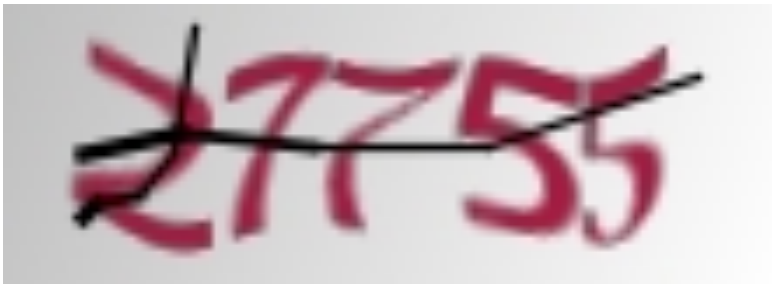


Figura A.2: CAPTCHA do TJMG.

A.1.4 Classificação

A classificação manual de CAPTCHAs é importante para possibilitar o treino de modelos preditivos. Para classificar um CAPTCHA você pode utilizar a função `classify()`, assim:

```
'imgs/tjmg/captcha2f6e4f7825d6.jpeg' %>%
  classify()
```

Essa função executa duas tarefas:

- Plota o CAPTCHA na tela.
- Abre um console para o usuário digitar o valor do CAPTCHA manualmente.

Ao escrever o valor o CAPTCHA e pressionar `<enter>`, a função `classify()` adicionará a classificação no nome do arquivo da imagem. A função `classify()` gera uma cópia para que seja impossível de perder a imagem original.

Algumas opções do `classify()`:

- `answers=` adicionar uma resposta ao invés de esperar abrir o console. Essa opção é útil quando as classficações são feitas automaticamente (e.g., por um quebrador de CAPTCHAs que usa o áudio no lugar da imagem.)
- `path=` colocar uma pasta para classificar os CAPTCHAs. Por padrão é a pasta onde os originais estão.

A.1.5 Carregar modelo

A função `load_model()` é responsável por carregar modelos pré treinados

```
modelo <- decryptr::load_model("tjmg")
modelo$model
```

A.1.6 Resolver captcha

A função `decrypt` resolve o CAPTCHA a partir de uma imagem e um modelo.

```
decrypt('imgs/tjmg/captcha2f6e4f7825d6.jpeg', modelo)
#> "27755"
```

Também é possível chamar `decrypt` com o nome do modelo no lugar do próprio modelo carregado.

Referências Bibliográficas

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Bursztein, E., Aigrain, J., Moscicki, A., and Mitchell, J. C. (2014). The end is nigh: Generic solving of text-based captchas. In *WOOT*.

Bursztein, E., Martin, M., and Mitchell, J. (2011). Text-based captcha strengths and weaknesses. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 125–138. ACM.

George, D., Lehrach, W., Kansky, K., Lázaro-Gredilla, M., Laan, C., Marthi, B., Lou, X., Meng, Z., Liu, Y., Wang, H., et al. (2017). A generative vision model that trains with high data efficiency and breaks text-based captchas. *Science*, 358(6368):eaag2612.

Golle, P. (2008). Machine learning attacks against the asirra captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 535–542. ACM.

Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. (2013). Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*.

Gray, R. M. (2006). Toeplitz and circulant matrices: a review. *Foundations and Trends in Communications and Information Theory*, 2(3):155–239.

LeCun, Y. et al. (2015). Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, page 20.

Mori, G. and Malik, J. (2003). Recognizing objects in adversarial clutter: Breaking a visual captcha. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE.

Motoyama, M., Levchenko, K., Kanich, C., McCoy, D., Voelker, G. M., and Savage, S. (2010). Re: Captchas-understanding captcha-solving services in an economic context. In *USENIX Security Symposium*, volume 10, page 3.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

Von Ahn, L., Blum, M., Hopper, N. J., and Langford, J. (2003). Captcha: Using hard ai problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311. Springer.

von Ahn, L., Blum, M., and Langford, J. (2002). Telling humans and computers apart automatically or how lazy cryptographers do ai. *Computer Science Department*, page 149.

Yan, J. and El Ahmad, A. S. (2008a). A low-cost attack on a microsoft captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 543–554. ACM.

Yan, J. and El Ahmad, A. S. (2008b). Usability of captchas or usability issues in captcha design. In *Proceedings of the 4th symposium on Usable privacy and security*, pages 44–52. ACM.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.