UNIVERSIDADE FEDERAL DE MINAS GERAIS
ENGENHARIA DE SISTEMAS

MACHINE LEARNING AUTOMATED MODEL COMPARISON: UM SOFTWARE PARA AUTOMATIZAR A COMPARAÇÃO ENTRE MÉTODOS DE APRENDIZADO

PAULO CIRINO RIBEIRO NETO

Orientador: Antônio de Pádua Braga
Universidade Federal de Minas Gerais

Coorientador: Gustavo Rodrigues Lacerda Silva
Universidade Federal de Minas Gerais

BELO HORIZONTE
NOVEMBRO DE 2017

PAULO CIRINO RIBEIRO NETO

MACHINE LEARNING AUTOMATED MODEL COMPARISON: UM SOFTWARE PARA AUTOMATIZAR A COMPARAÇÃO ENTRE MÉTODOS DE APRENDIZADO

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Sistemas da Universidade Federal de Minas Gerais, como requisito parcial para aprovação na disciplina de Trabalho de Conclusão de Curso I

Orientador: Antônio de Pádua Braga
Universidade Federal de Minas Gerais

Coorientador: Gustavo Rodrigues Lacerda Silva
Universidade Federal de Minas Gerais

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ENGENHARIA DE SISTEMAS
BELO HORIZONTE
NOVEMBRO DE 2017

AGRADECIMENTOS

Agradeço ao meu orientador Prof. Dr. Antônio de Pádua Braga, pelo apoio ao longo dessa trajetória. Aos colegas do LITC Dr. Luiz Carlos Bambirra Torres, Dr. Frederico Gualberto Ferreira Coelho e, especialmente, ao meu coorientador Gustavo Rodrigues Lacerda. Aos meus colegas de sala, professores e especialmente ao secretário do curso de Engenharia de Sistemas Júlio César Pereira de Carvalho. Por fim, gostaria de mostrar o meu reconhecimento à minha família, pois acredito que sem o apoio deles seria muito difícil vencer esse desafio.

“The only excuse for making a useless thing is that one admires it intensely.” (Oscar Wilde, prefácio, O retrato de Dorian Gray)

RESUMO

O tema desse trabalho de conclusão de curso é a criação de um software para automatizar etapas do processo de *benchmarking* de modelos de aprendizado de máquina.

Foi feito um software, escrito na linguagem de programação R, capaz de comparar modelos de aprendizado de máquina disponíveis na literatura com modelos criados pelo usuário. O software faz o carregamento das bases de dados padrões, suas chamadas, as métricas de avaliação e os testes estatísticos de qualidade para os problemas de classificação e regressão.

Será discutido nesse trabalho, o impacto social de software livre e os aspectos técnicos e definições matemáticas do aprendizado supervisionado. Por fim será apresentado o pacote e um exemplo de como utiliza-lo na função de auxiliar o *benchmarking* entre os modelos.

Palavras-chave: Software Livre. Aprendizado de Máquina. Linguagem R. Teste de Modelos.

ABSTRACT

The theme of this work is the creation of a software capable of automating parts of machine learning models benchmarking.

A software capable of comparing established machine learning model with ones created by the user was made in this project. The software loads databases, create function calls for testing, implements avaliation functions and defines statistical tests routines for quality mesures in classification ajd regression problems.

The discussion of social impact of free software and the technical and mathematical aspects of supervised learning, is also part of the project. And, in the end the software is presented along with a demo code.

Keywords: Free Software. Machine Learning. R programming Language. Model Testing. Benchmarking.

LISTA DE FIGURAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Program Interface*
GNU	*GNU's Not Unix*
GPL	*General Public License*
MIT	*Massachusetts Institute of Technology*
BSD	*Berkeley Software Distribution*
NSA	*National Security Agency*
CRAN	*Comprehensive R Archive Network*
CPU	*Central Processing Unit*
GPU	*Graphical Processing Unit*
HTTP	*Hypertext Transfer Protocol*
RNA	Redes Neurais Artificiais
KNN	*K Nearest Neighbours*
RNN	*Rede Neural Artificial*
MLP	*Multilayer Perceptron*
SVM	*Support Vector Machines*

LISTA DE SÍMBOLOS

x	Entrada real
y	Saída real
$\hat{y}$	Saída estimada
$p(x)$	Distribuição de probabilidade de uma variável x
$p(y\|x)$	Probabilidade condicional de x dado y
μ	Média
σ	Desvio padrão
ϵ	Erro aleatório com $\mu = 0$
ω	Parâmetro de uma função
Ω	Conjunto de todos os parâmetros de uma função
$\mathbb{R}$	Conjunto dos números reais
R^+	Conjunto dos números reais positivos
$\mathbb{I}$	Conjunto dos números inteiros
$\mathbb{N}$	Conjunto dos números naturais
L	Função de Custo
f	Função do Aprendizado de Máquina
g	Função do Sistema
$d(r, s)$	Distância entre as variáveis r e s
$\in$	Pertence
$\wedge$	Simbolo de *e* lógico
$\vee$	Simbolo de *ou* lógico
$\rightarrow$	Implica

SUMÁRIO

Capítulo 1

Introdução

1.1 Justificativa

Um dos principais objetivos de softwares é auxiliar os usuários nos trabalhos com sistemas computacionais. Uma das formas de atingir esse objetivo é automatizando tarefas rotineiras, repetitivas, desinteressantes e que consomem muito tempo, liberando as pessoas para tarefas mais importantes (ZAMBIASI; RABELO, 2012).

Uma parte fundamental no ciclo do desenvolvimento de novos algoritmos em aprendizado de máquina é o processo de *benchmarking*. Esse procedimento consiste na comparação do novo método com outros modelos, além, é claro, de testes estatísticos que permitam extrair conclusões quantitativas e qualitativas.

Realizar *benchmarking* pode se tornar uma tarefa repetitiva, trabalhosa e consequentemente cara. O projetista precisa implementar todos os métodos padrões, baixar e pré-processar todas as bases de testes, definir o planejamento e análise dos experimentos, e por fim, realizar testes estatísticos e visualizações de dados para que seja possível extrair suas conclusões.

Dessa forma, a automatização da tarefa de *benchmarking* seria benéfica ao projetista no sentido de economizar tempo para realizar tarefas mais importantes. Além disso, uma rotina de teste padronizada é interessante pois faz com que seja possível a comparação de estudos feitos separadamente.

1.2 Objetivo

Esse trabalho tem como objetivo a criação de um software, que seja capaz de automatizar a tarefa de *benchmarking* para o teste de novos modelos de aprendizado de máquina.

O projeto será executado na forma de pacote aberto, licença GNU GPLv3, da linguagem de programação estatística R. O pacote será implementado utilizando a própria linguagem R, sua API para C e $C++$ e outros pacotes livres feitos pela comunidade.

Ao fim do ciclo de desenvolvimento, o pacote deverá ser capaz de comparar modelos para tarefas de classificação e regressão desenvolvidos pelos usuários com outros modelos já presentes no pacote. O usuário poderá fornecer as bases de dados de testes ou utilizar as bases do pacote. O pacote deverá fornecer também a opção de testes estatísticos e visualizações dos resultados.

O objetivo é também ajudar a comunidade com o projeto, dessa forma ele foi enviado ao repositório CRAN.

1.3 Organização do trabalho

O trabalho está estruturado de forma que o capítulo 2 tem como objetivo contextualizar e discutir os aspectos sociais e culturais do software livre, bem como falar brevemente da história desse movimento e das suas definições de liberdades sociais.

O Capítulo 3, discutirá sobre a linguagem de programação R, como ela está inserida dentro do movimento de software livre e como ela está organizada.

Os capítulos 4 e 5 discutirão, de forma sucinta, os conceitos de aprendizado de máquina e de estatísticas de testes que são as bases da implementação do software.

Por fim, o capítulo 6 abordará a apresentação do projeto de software, e é feito um exemplo de como utiliza-lo.

Nesse trabalho não serão discutidas as bases de dados disponíveis no pacote, e tampouco as funções implementadas nele. Esses assuntos são tratadas na documentação do software que se encontra no apêndice A.

Capítulo 2

Software livre

2.1 Definição de Software Livre

A definição de software livre apresenta os critérios para determinar se um programa de software é qualificado como livre. Essa definição pode mudar conforme o momento histórico, mas atualmente é definida pela GNU como : *software que os usuários têm a liberdade de executar, copiar, distribuir, estudar, alterar e melhorar* (INC, 2012) .

Nesse contexto, o conceito de livre diz respeito à "liberdade de expressão", e não à gratuidade (WILLIAMS; STALLMAN, 2010). Com essas liberdades, os usuários, tanto individualmente quanto coletivamente, controlam o programa e o que ele pode fazer. Quando os usuários não controlam o programa, chamamos ele de programa *não livre* ou *proprietário* (INC, 2012).

Em linhas gerais, um software livre deve, obrigatoriamente, obedecer a quatro liberdades (INC, 2012; WILLIAMS; STALLMAN, 2010; LESSIG; STALLMAN, 2002) :

1. A liberdade de executar o programa como desejar, para qualquer propósito;
2. A liberdade de estudar como funciona o programa e mudá-lo para que ele faça a sua computação como você deseja;
3. A liberdade de redistribuir cópias para que você possa ajudar seu vizinho;
4. A liberdade de distribuir cópias de suas versões modificadas para outros.

Naturalmente, essas liberdades estão relacionadas às informações divulgadas entre desenvolvedor e usuário. A liberdade **2**, por exemplo, implica na necessidade de o desenvolvedor divulgar abertamente o código fonte de um software e permitir que esse seja modificado (INC, 2012). Além disso, é obvio que para os usuários terem a liberdade de decidir sobre sua computação, o software livre, por sua vez, não pode utilizar nenhum código *não livre*.

É importante notar que não existem, nessas liberdades, quaisquer limitações relacionadas

ao uso comercial do software. Isso significa que empresas podem cobrar pelo uso do software em versões modificadas ou não.

Além das liberdades básicas de um software livre, existem as licenças de uso. Cada licença específica, restringe ou garante mais liberdades sobre o software, sem afetar as quatro fundamentais. De forma forma geral, existem quatro famílias de licenças : *Permissiva*, *Fracamente Protetiva*, *Fortemente Protetiva* e *Protetiva de Rede* (WILLIAMS; STALLMAN, 2010).

2.2 História do software livre

O surgimento do movimento de software livre está altamente atrelado à academia e ao desenvolvimento dos sistemas operacionais UNIX, GNU e Linux.

O UNIX tem suas origens na *joint venture*, lançada no final da década de 1960 pela *Bell Labs* e MIT para criar um novo sistema operacional chamado *Multics* (TOZZI; ZITTRAIN, 2017). Utilizando o conhecimento adquirido nesse projeto, alguns dos programadores desenvolveram, paralelamente, um sistema operacional para oferecer mais flexibilidade aos usuários, nomeado UNIX.

Em 1975, Ken Thompson juntamente com Bill Joy e Chuck Haley, começaram a distribuir uma versão *open source* do UNIX chamada BSD. No ano seguinte, o lançamento de uma edição revista foi denominada 2BSD (TOZZI; ZITTRAIN, 2017).

No ano de 1984, o programador Richard Stallman fundou o Projeto GNU. A GNU GPL permitia aos usuários modificar o código e distribuir a versão melhorada sob a mesma licença. O sistema operacional GNU não tinha um *kernel*, até que em 1991, Linus Torvalds desenvolveu o *kernel* do Linux que foi integrado no sistema operacional GNU em 1992 (TOZZI; ZITTRAIN, 2017).

Nos anos seguintes, foram introduzidas versões, comerciais e aprimoradas, do sistema operacional Linux por fornecedores como *Red Hat*, *Mandriva* e *Novell*.

Com a criação de sistemas operacionais que poderiam ser utilizados com total liberdade, surgiu então uma demanda por *software* livres que funcionassem nesses ambientes. Da mesma forma que a comunidade se juntou para aprimorar a base do *kernel* do Linux, eles se juntaram e fizeram os mais diversos *software* para atender essa demanda.

Nos dias atuais existem inúmeras comunidades que fazem os mais variados tipos de softwares utilizando os mesmos princípios criados por Richard Stallman. Os *software* livres difundiram na sociedade, e hoje são peças fundamentais para infraestrutura computacional, periféricos, celulares e virtualmente qualquer outro dispositivo computacional.

2.3 A importância social do software livre

Atualmente, os softwares livres são fundamentais principalmente em três áreas sociais: à proteção da liberdade individual, ao avanço da computação e à acessibilidade da educação e conhecimento.

2.3.1 Software livre à favor da proteção da liberdade individual

Uma celebre frase da comunidade de software livre diz, 'Os softwares não livres, onde o usuários não controlam o programa, o programa controla os usuários' (WILLIAMS; STALLMAN, 2010).

Essa frase, resume uma preocupação crescente com o software proprietário, a de que sempre há alguma entidade, que controla o programa e através dele, exerce poder sobre seus usuários.

Esse poder é enxergado por alguns como uma afronte ao direito de privacidade do indivíduo. Hoje, serviços de busca e redes sociais, utilizam dos dados de navegação para gerar propagandas sob medidas. Muitas pessoas, consideram a forma que essas empresas manipulam as informações como uma forma de censura e venda de informação confidencial.

Em alguns casos, empresas que desenvolvem softwares proprietários foram ligadas a escândalos onde propositalmente construíram *backdoors* em seus produtos que dão acesso a informações sem as devidas permissões dos usuários. Um exemplo é o caso do *Kindle*, que possui uma *backdoor* que permite apagar livros (GNU Operating System, 2017) .

Um outro senário onde é importante que o software seja livre, é para proteger os usuários de acesso externo indesejado. Um exemplo disso é o caso do ex analista da NSA (TATE et al., 2013), Edward Joseph Snowden, que em junho de 2013, revelou como a agência americana utilizava de falhas de seguranças, propositais ou não, para espiar na população mundial.

O principio que o software livre protege a liberdade Individual, contra empresas mau intencionadas ou governos abusivos, é que quando o código de um programa é aberto, a comunidade pode ver oque ele faz e testar todas as falhas que o mesmo possa ter (GNU Operating System, 2017).

2.3.2 Software livre à favor do avanço da computação

Após a construção dos sistemas operacionais livres, surgiram varias distribuições e variações, cada qual para atender um nicho. Esses avanços, tornaram possível que hoje, os sistemas operacionais baseados no Linux e UNIX dominassem o setor de infra estrutura

computacional. Possibilitando que empresas e órgãos governamentais customizassem esses *software* para criar soluções específicas para suas necessidades, que por sua vez não são necessariamente livres.

Essa abordagem se tornou tão prática que, dados da W3Techs (2017) e Top 500 project (2017), mostram que esses sistemas operacionais são utilizados em 66.8% de todos os servidores públicos de internet e 99.88% dos supercomputadores do mundo.

A ultima grande plataforma que popularizou a utilização do Linux foi o sistema operacional Android. Construído inicialmente para ser um software de telefones celulares, esse sistema operacional criado com base no *kernel* do Linux, se espalhou pelos mais diversos aparelhos, como televisões, *tablets* e até mesmo geladeiras (RILEY, 2012). Esse software se popularizou tanto que, segundo o CEO da Google, Sundar Pichai, é utilizado em mais de 2 bilhões de dispositivos ativos (RILEY, 2012).

Além dos avanços em sistemas operacionais, o movimento de software livre alavancou o desenvolvimento comunitário de softwares de computação livres. Um exemplo disso é a *Apache Software Foundation*, que é uma corporação americana sem fins lucrativos formada por uma comunidade descentralizada de desenvolvedores de código aberto.

Os projetos Apache são feitos em desenvolvimento colaborativo, baseado em consenso e uma licença de software aberta e pragmática. Cada projeto é gerenciado por uma equipe de especialistas técnicos auto-selecionados que são contribuidores ativos para o projeto.

O projeto inicial da Apache foi o *HTTP Server*, que era um sistema para processar protocolos web básicos na internet. Contudo, hoje são 315 projetos nas mais diversas áreas da computação, e incluem *software* de *Big Data* como o *Spark* e *Hadoop*, gerenciamento de projetos como o *Maven* e até mesmo software de escritório como o *Open Office*.

2.3.3 Software livre à favor da acessibilidade da educação e conhecimento

As escolas e universidades, influenciam o futuro da sociedade através do que ensinam. Ensinar um programa proprietário é implantar a dependência de um artifício que não é de sua propriedade. Isso diminui a capacidade do futuro profissional em exercer os conhecimentos que lhe foram ensinados (Richard Stallman, 2017).

A utilização de software livre como ferramenta de ensino, empodera os estudantes a utilizarem os conhecimentos técnicos na vida após a universidade (LESSIG; STALLMAN, 2002). Utilizando software livre, um profissional é capaz de fazer uso de programas como ferramenta básica de trabalho gratuitamente, ou ainda utilizar soluções livres como base para criar um produto próprio.

Além de ser útil para o estudante, os *software* livres são importantes para o avanço da pesquisa acadêmica na universidade. A comunidade de *software* livre têm em suas liberdades básicas, a liberdade de estudar como um programa funciona e permitir que os usuários melhorem e redistribuam esse programa. Este espírito de comunidade permite que pesquisadores em locais diferentes do mundo, sem quaisquer dificuldades, compartilharem suas pesquisas e conhecimentos.

Além de promover o compartilhamento da informação, os softwares livres também promovem uma acessibilidade universal dos avanços técnico-científicos, uma vez que permitem pesquisadores e empresas de ponta à compartilhar seus resultados e códigos com o resto do mundo (LESSIG; STALLMAN, 2002). Isso permite que mesmo pesquisadores com recursos escassos, façam aplicações ou pesquisa com esses avanços .

Um exemplo disso, é o caso do software *Tensor Flow* (ABADI et al., 2016), feito pela Google. Esse é um programa extremamente complexo que funciona como motor de operações numéricas, utilizando a abstração de computação em grafos de forma escalável para CPU's e GPU's. Além de compartilhar o código do *Tensor Flow* com a comunidade, a empresa também disponibilizou inúmeros modelos de RNA, como a *LeNet* (SZEGEDY et al., 2015) que é um modelo treinado para identificar objetos em imagens. Utilizando esse avanço, pesquisadores do mundo inteiro foram capazes de utilizar esses modelos em suas próprias pesquisas.

Capítulo 3

Linguagem de Programação R

3.1 História da linguagem R

R é uma linguagem e ambiente para computação estatística e gráfica, que foi criada na década de 90 por Ross Ihaka e Robert Gentleman, enquanto ambos trabalham na Universidade de Auckland(WICKHAM, 2015). Esse é um projeto GNU que é semelhante a linguagem e ao ambiente *S* desenvolvida na *Bell Laboratories* por John Chambers.

O *R* pode ser considerado como uma implementação diferente da linguagem de programação *S*, ao ponto que código escrito em *S* pode ser executado de forma inalterada pelo interpretador de *R*. Contudo, a principal diferenças entre os dois ambientes, é que o *R* é um projeto de *software* livre sob licença GPL GNU e o *S* possui licença proprietária(WICKHAM, 2015).

Atualmente esse ambiente é utilizando principalmente nas áreas de estatística e análise de dados. O sucesso da linguagem nessas tarefas pode ser explicado pela grande variedade de algoritmos em modelagem linear e não-linear, testes estatísticos clássicos, análise de séries temporais e técnicas gráficas. Nos dias atuais, a linguagem é utilizada na indústria e academia, de forma que já é a sexta linguagem de programação mais popular (CASS, 2017).

Apesar de ser uma linguagem livre e aberta, o desenvolvimento e manutenção do ambiente é controlada pela *R Foundation* e pelo grupo de 20 curadores chamados de *R core team*. Contudo, qualquer pessoa pode contribuir com o avanço da linguagem por meio de pacotes, tradicionalmente disponibilizados no repositório oficial CRAN e divulgados na revista chamada *The R Journal*.

3.2 Organização da linguagem de programação *R*

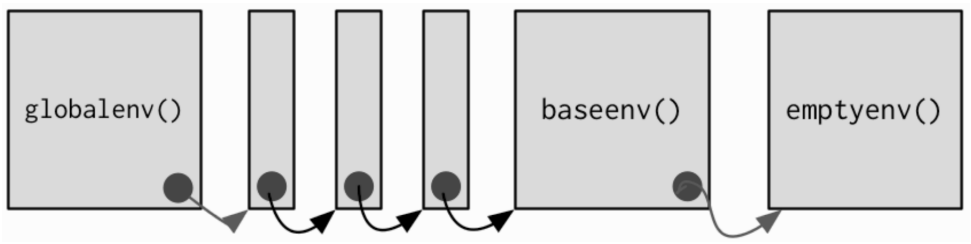
O *R* é uma linguagem de programação que não foi planejada para ser tão versátil. Inicialmente, foi criado um interpretador que executava comandos da linguagem *S*, onde o ambiente era composto por funções de matemáticas, estatísticas, gráficas e de leitura de dados (R-PROJECT, 2017).

Como a linguagem foi criada por estatísticos para estatísticos, não foram definidos padrões de programação. Dessa forma ela é considerada uma linguagem de quarta geração, ou seja, uma linguagem de domínio (R-PROJECT, 2017).

Possui suporte para os principais paradigmas de programação, incluindo os modelos declarativo, imperativo, orientado à objeto, procedural, funcional e outros. É um linguagem naturalmente lenta, por conta de ser interpretada e concorrente. Entretanto, fornece interface para conectar com linguagens de alta performance, *C*, *C++* e *Fortran*, e programação paralela pela plataforma *OpenMP* . O código do interpretador *R* possui mais de 750 mil linhas, das quais 39% são escritas em *C*, 27.1% em *Fortran*, 19.7% em *R*, 8.1% em *Autoconf* e 2% em *shell* (OPENHUB, 2017).

Esse interpretador foi definido para trabalhar com o conceito de *ambientes*, que são grupos de funções e variáveis organizadas por precedência de contexto. O formato dos contextos podem ser observados na figura 1 .

Figura 1 – *Ambientes* da linguagem de programação R



Fonte: Wickham (2015)

Em linhas gerais, o ambiente onde o usuário trabalha é chamado de *globalenv*, as funções da linguagem *R* estão definidas no ambiente de *baseenv* e os gerenciadores de memória, o coletor de lixo e os demais controladores do próprio interpretador estão no ambiente *emptyenv*.

A linguagem *R*, interpreta as funções de forma a percorrer os ambientes hierarquicamente. Quando o usuário chama o interpretador, ele inicialmente procura as funções no *globalenv*, quando não encontra, ele busca recursivamente nos ambientes *pai* (The R Foundation for Statistical Computing, 2014).

Todos os ambientes entre o *globalenv* e o *baseenv*, são chamados de *ambientes de pacote*. O ultimo pacote a ser invocado é chamado de pai do *globalenv*, e é filho do penúltimo pacote invocado.

Esses pacotes foram a forma encontrada pelos desenvolvedores da linguagem para permitir que a comunidade contribuísse com o avanço da linguagem, sem afetar as funções base. Além disso, esse processo recursivo de busca de *ambientes* é o que permite os pacotes da linguagem R utilizarem uns aos outros (The R Foundation for Statistical Computing, 2014).

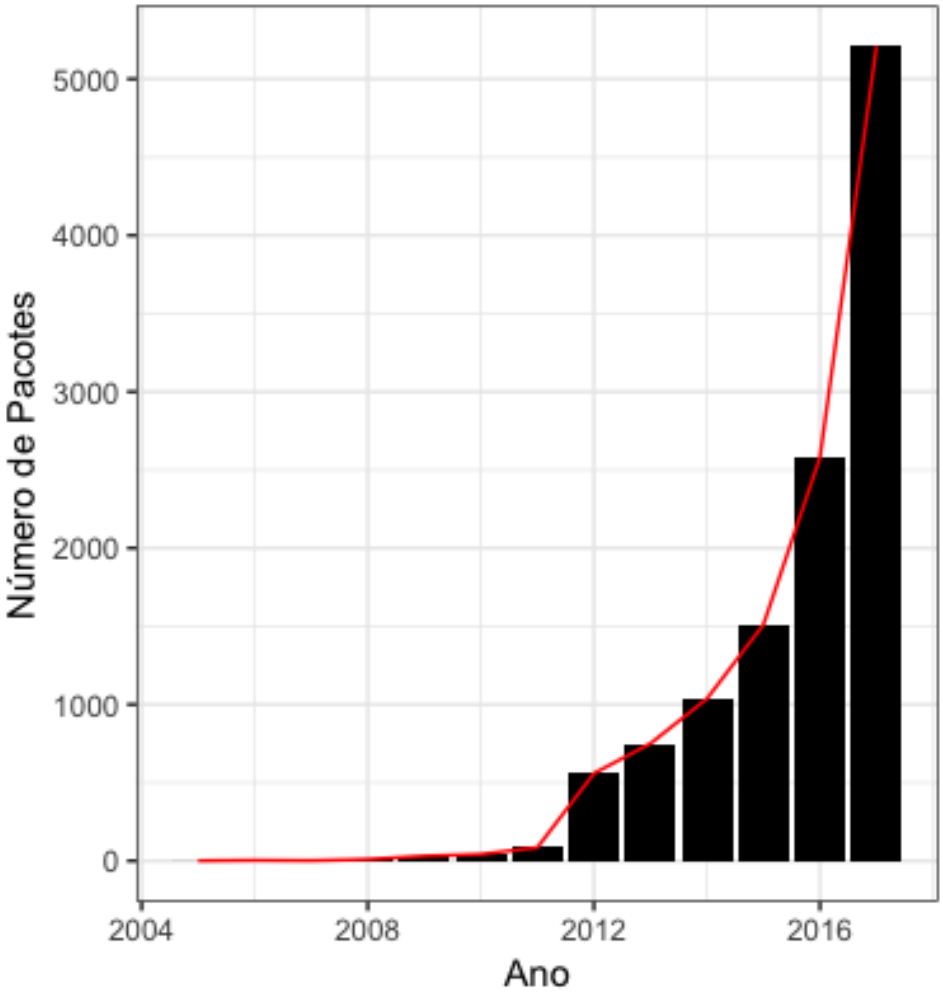
3.3 A comunidade R e seus pacotes

A linguagem R surgiu dentro do meio universitário como uma alternativa à *software* proprietários. Dessa forma, a expansão da comunidade de usuários ocorreu inicialmente da mesma maneira, no âmbito acadêmico e por meio *software* e pacotes livres. Contudo, hoje o R possui grande aporte de empresas privadas, como Google e AT&T (The R Foundation for Statistical Computing, 2014).

Atualmente o repositório CRAN possui mais de 11800 pacotes, que implementam modelos dos mais variados, incluindo mas não limitado à gráficos, biológicos, estatísticos, ecológicos e de aprendizado de maquina. É importante ressaltar que todos os pacotes devem ter documentação, código aberto e possuírem licença livre (The R Foundation for Statistical Computing, 2014).

Esses padrões definidos pela comunidade têm se mostrado extremamente efetivos, no sentido da popularização da criação de pacotes, vide figura 2.

Figura 2 – Número de pacotes disponíveis no CRAN ao longo do tempo



Fonte: https://cran.r-project.org/web/packages/available_packages_by_date

Capítulo 4

Aprendizado de máquina

4.1 Formulação e caracterização do aprendizado de máquina

Nos primórdios da inteligência artificial, pesquisadores foram capazes de resolver problemas que são intelectualmente difíceis para os seres humanos, mas relativamente simples para os computadores. Exemplo disso são os problemas que podem ser descritos por uma sequência de regras ou que podem ser solucionados por operações matemáticas (GOODFELLOW; BENGIO; COURVILLE, 2016).

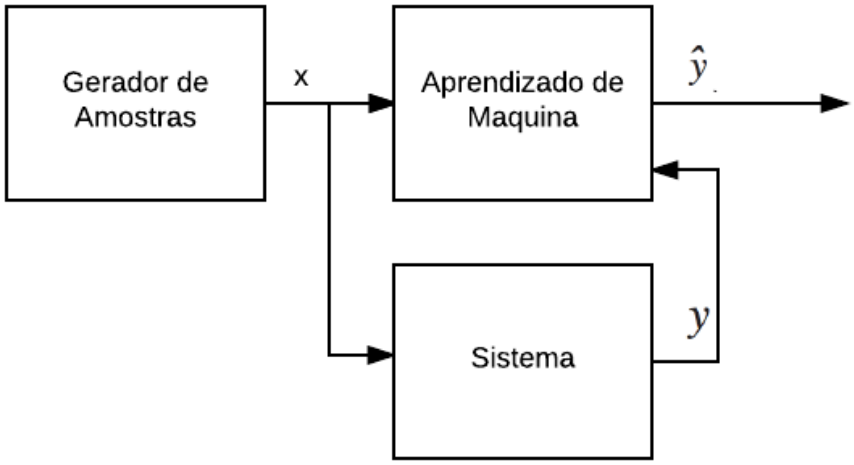
A dificuldade surgiu para fazer com que os computadores fossem capazes de resolver tarefas fáceis para os humanos, mas difíceis de serem descritas de forma algorítmica, como reconhecer escrita, identificar rostos em imagens ou ainda definir regras de decisão de forma autônoma (GOODFELLOW; BENGIO; COURVILLE, 2016).

Eis que surgiu o campo de Aprendizado Máquina, uma alternativa que permitia computadores aprender por exemplo e não por regras. Ao reunir o conhecimento da experiência, esta abordagem evita a necessidade de operadores humanos especificar formalmente todo o conhecimento que o computador precisa. Isso ocorre através da combinação do aprendizado de conceitos simples, em formato pré-definido, combinados de forma hierárquica para formar um aprendizado de conceitos complicados (GOODFELLOW; BENGIO; COURVILLE, 2016).

Segundo Cherkassky e Mulier (2007), o processo de aprendizado é definido como a estimação de uma relação ou estrutura, previamente desconhecida, entre entrada e saída. Segundo o autor, o processo de aprendizado normalmente envolve três componentes 3 : o *Gerador* de amostras, um *Sistema* que gera as saídas reais e o *Aprendizado de Máquina* que estima a relação desconhecida entre entrada e saída.

Cherkassky e Mulier (2007) explica que o *Gerador* produz aleatoriamente, com distribuição de probabilidade $p(x)$, vetores $x \in \mathbb{R}^n$, que são as características. O *Sistema* é capaz de

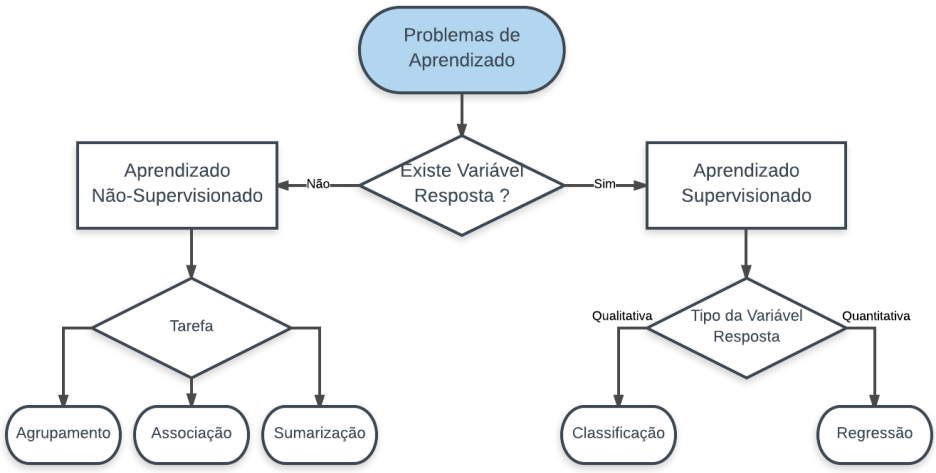
Figura 3 – Formulação do Aprendizado



produzir, com probabilidade $p(y|x)$, saídas $y = g(x) + \epsilon$ em que ϵ é um ruido branco de $\mu = 0$. De forma geral, o *Aprendizado de Máquina* é capaz de implementar uma função $\hat{y} = f(x, \omega)$, $\omega \in \Omega$, em que Ω é conjunto de parâmetros da função f.

O aprendizado de máquinas clássico diz respeito à aprender pelo exemplo, que é definido como um conjunto de dados, previamente obtidos pelo *Gerador* e *Sistema*, composto por variáveis quantitativas e qualitativas que são características do problema. Na situação onde existe uma variável resposta, o aprendizado é chamado de supervisionado, na situação onde essa variável não existe, o problema é definido como aprendizado não supervisionado. A figura 4 ilustra essa taxonomia, bem como algumas de suas subdivisões.

Figura 4 – Principais Problemas em Aprendizado de Máquina



4.2 Aprendizado supervisionado

No aprendizado supervisionado, o objetivo é prever o valor de uma variável resultado com base em variáveis características. Na situação que a variável resposta é quantitativa, o problema é considerado como uma tarefa de regressão. Quando essa variável resposta é qualitativa, a tarefa é chamada de classificação.

De acordo com Cherkassky e Mulier (2007), existem duas interpretações para o problema de aprendizado : identificação e imitação. A interpretação escolhida para fundamentar esse trabalho é a de imitação, descrita por Vapnik e Chervonenkis (1971).

O objetivo do aprendizado é encontrar uma função $f(x, \omega)$ que aproxima, da melhor forma possível, a saída y do *Sistema*. Para formular esse problema matematicamente, Cherkassky e Mulier (2007), assumem pares de características e respostas, (x_i, y_i) em que $i = (1, 2, ..., n)$ e $n \in \mathbb{N}$, e uma função de custo $L(y, f(x, \omega))$que mede a discrepância entra saída do *Sistema* y e do *Aprendizado de Máquina* $\hat{y}$. Então, formulam que a tarefa do aprendizado é descrita pela equação 1 .

$$\arg\min_{f} L(y, f(x, \omega)) \tag{1}$$

A principal diferença entre problemas de classificação e regressão é a função de custo. Como a variável de saída de cada um desses problemas tem um formato diferente, a função de custo deve ser adaptada especificadamente para ele. Contudo, a formulação do aprendizado feito na equação 1 permanece inalterada, pois o objetivo é sempre minimizar a discrepância entre *Sistema* e *Aprendizado de Máquina* (CHERKASSKY; MULIER, 2007).

4.2.1 Problemas de classificação

A situação mais simples de classificação é aquele em que a saída y assume apenas dois valores, $y = 0$ ou $y = 1$, em que é chamada de problema de classificação binária. Nessa situação, segundo Hastie, Tibshirani e Friedman (2009), a função de custo específica é do formato 2.

$$L(y, f(x, \omega)) = \begin{cases} 0, & \text{se } y = f(x, \omega) \\ l, & \text{se } y \neq f(x, \omega) \end{cases} \quad l \in \mathbb{R}^+ \tag{2}$$

O senário em que y assume q valores, em que $q > 2 \wedge q \in \mathbb{N}$, pode ser simplificado em q problemas de classificação binária em que o problema j tem forma $y = j \rightarrow y_j = 1$ e $y_j \neq\rightarrow y_j = 0$, para $0 \leq j \leq q \wedge j \in \mathbb{N}$.

4.2.2 Problemas de regressão

O problema de regressão é caracterizado, segundo Hastie, Tibshirani e Friedman (2009), por $y \in \mathbb{R}$, dessa forma a função de custo normalmente é formulada como mostra a equação 3, onde d é uma métrica genérica de distância entre dois números reais.

$$L(y, f(x, \omega)) = d(y, f(x, \omega)) \tag{3}$$

4.3 Métodos de Classificação

Nesse capítulo serão abordados os métodos de aprendizados implementados no pacote MLAT (NETO, 2018).

4.3.1 *k Nearest Neighbours*

O método *k Nearest Neighbours* (**kNN**) é o algoritmo não paramétrico de classificação mais simples e conhecido (JAMES et al., 2013) . Seu funcionamento se dá de forma que a predição de uma amostra desconhecida é a moda das k amostras mais próximas. De forma genérica sua implementação pode ser feita utilizando o pseudocódigo abaixo adaptado do trabalho de Tay, Hyun e Oh (2014) :

Algoritmo 1: Algoritmo k-*Nearest Neighbours*

Input: x amostra desconhecida, k é o parâmetro do número de vizinhos, $\{\mathbb{X}, \mathbb{Y}\}$ é o conjunto de pares amostras e respostas conhecidas
Output: a predição da amostra desconhecida y
for *each* $x_i \in \mathbb{X}$ **do**
  Computa a distância da amostra conhecida i para a desconhecida $\mathbb{D}_{\sqsupset} = d(x, x_i)$
end
Seleciona o conjunto $\mathbb{L}$ referente as k respostas com menor distância $\mathbb{D}$, onde $\mathbb{L} \in Y$
Computa a resposta $y = moda(\mathbb{L})$

É importante ressaltar algumas características do kNN, como sua simplicidade de implementação e a não existência de uma etapa de treinamento. Além disso, o método é capaz de produzir superfices de decisão não lineares e é robusto, dado uma escolha adequada do parâmetro k, a conjuntos ruidosos.

Entretanto esse algoritmo se baseia na memória das amostras conhecidas, fazendo com que ele não seja capaz de generalizar sua superfície de decisão para novas amostras fora do espaço conhecido. Outra consequência de ser um sistema baseado em memória é sua complexidade $O(n)$, para espaço e tempo, no momento de predição.

4.3.2 *Support Vector Machines*

As *Support Vector Machines*, em português Máquinas de Vetor de Suporte, foram inicialmente propostas em Boser, Guyon e Vapnik (1992). Esse método foi produzido como uma alternativa que, diferente de outros algoritmos tradicionais, não minimiza o erro do classificador, mas, sim maximiza a margem de separação entre duas classes (JAMES et al., 2013). O intuito disso é produzir um classificar que seja capaz de encontrar uma superfície de decisão mais genérica, e consequentemente, que tenha melhor desempenho em amostras desconhecidas (CORTES; VAPNIK, 1995).

A formulação inicial dessa metodologia sofreu adaptações e melhorias ao longo do tempo, em Boser, Guyon e Vapnik (1992) o classificador possuía margem rígida e assumia que existia uma superfície que separa perfeitamente as classes, chamado hoje de *Hard Margin SVM*.

Dado um conjunto de pares (x_i, y_i) é possível formular esse classificador conforme a equação 4 (JAMES et al., 2013). Nessa formulação as respostas são $y_i \in \{1, -1\}$, $\beta_0 + \sum_{j=1}^{p} \beta_i x_{ij}$ é o hiperplano de separação e M é a variável de otimização que diz respeito ao tamanho da margem.

$$\begin{aligned} &\underset{\beta_0,\beta_1,...,\beta_p,M}{\text{Maximizar}} : M \\ &\text{Sujeito a :} \\ &\sum_{j=1}^{p} {\beta_j}^2 = 1, \\ &y_i(\beta_0 + \sum_{j=1}^{p} \beta_i x_{ij}) \geq M, \forall i = 1, ..., n \end{aligned} \tag{4}$$

Em Cortes e Vapnik (1995) é demostrada uma alteração na formulação original, fazendo ser possível encontrar uma solução mesmo em problemas que as amostras não sejam completamente separáveis. Isso é feito por meio da introdução de um parâmetro ϵ_i, que diz respeito a penalidade de amostras classificadas erroneamente. Essa alteração na formulação, conhecida como *Soft Margin* SVM, é demonstrada pela equação 5 (JAMES et

al., 2013).

$$\underset{\beta_0,...,\beta_p,\epsilon_1,...,\epsilon_n,M}{\text{Maximizar :}} \; M$$

Sujeito a :

$$\begin{aligned} &\sum_{j=1}^{p} {\beta_j}^2 = 1, \\ &y_i(\beta_0 + \sum_{j=1}^{p} \beta_i x_{ij}) \geq M(1-\epsilon_i), \forall i = 1, ..., n, \\ &\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \geq C \end{aligned} \tag{5}$$

As equações 4 e 5 definem modelos de aprendizado com uma superfície de decisão linear. Para solucionar essa questão foi introduzido, em Boser, Guyon e Vapnik (1992) , o conceito conhecido como *Kernel Trick*.

A implementação do *Kernel Trick* é feita substituindo o produto interno, $f(x) = \beta_0 + \sum_{j=1}^{p} \beta_i x_{ij}$ por uma função genérica $K(x_i)$ não linear (JAMES et al., 2013). Essa função pode introduzir termos de ordem superior ou até mesmo aplicar funções não lineares, tudo com o objetivo de facilitar a separabilidade das amostras.

4.3.3 *Multilayer Perceptron*

Uma rede neural artificial é formada por um conjunto de nodos, chamados de neurônios, com capacidade de processamento local e independe agrupados em uma topologia definida (BRAGA A. P., 2007). O modelo *Multilayer Perceptron* (MLP) é um tipo de rede neural artificial *feedfoward* que utiliza em sua camada intermediária neurônios perceptron (ABADI et al., 2016).

$$y = \phi(\sum_{i=1}^{m} x_i w_i + b) \tag{6}$$

O neurônio perceptron, definido pela equação 6, é o resultado da combinação linear dos termos das entradas x_i ponderados por pesosw_i e um bias b após a introdução de uma não linearidade pela função de ativação ϕ. A função de ativação ϕ pode assumir formatos diferentes, todavia, tradicionalmente ela é logística, tangente hiperbólica ou RELU conforme equações 7, 8 e 9 respectivamente (GOODFELLOW; BENGIO; COURVILLE, 2016).

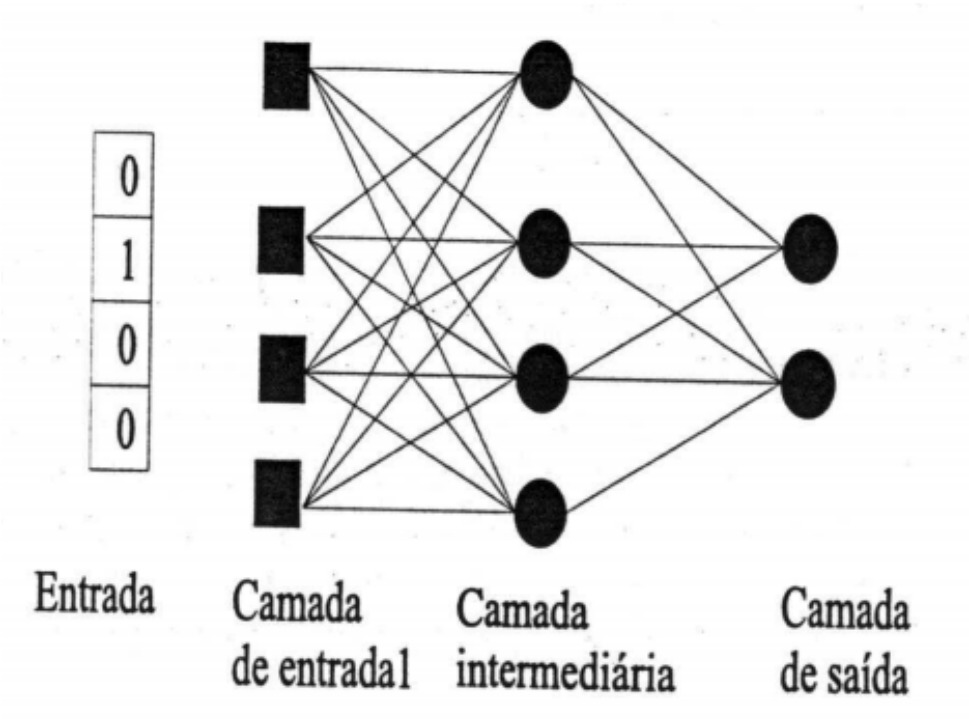
$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{7}$$

$$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{8}$$

$$\phi(z) = max(z, 0) \tag{9}$$

A topologia da MLP, ilustrada na imagem 5, é compostas por 3 tipos de camadas : entrada, intermediária ou escondida, e, saída (BRAGA A. P., 2007). A rede possui uma camada de entrada com tamanho definido pelo número de variáveis do espaço dos dados de entrada, ela pode possuir uma ou mais camadas intermediária, formadas por neurônios perceptron, de tamanhos parametrizados por seu projetista.

Figura 5 – Formulação do Aprendizado



(BRAGA A. P., 2007)

A camada de saída em um problema de classificação também pode assumir formatos diferentes, para um problema binário tradicionalmente é utilizado um neurônio do tipo logística conforme equação 7. No caso de problemas de classificação em que a saída pode assumir mais de 2 valores a camada de saída deve possuir o mesmo número de neurônios que valores de saída possíveis. Caso o projetista deseje ainda uma saída probabilística é ainda possível utilizar uma camada final do tipo *softmax* conforme equação **??**, em que k representa cada neurônio da camada de saída (GOODFELLOW; BENGIO; COURVILLE,

2016).

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{10}$$

4.4 Métodos de Regressão

4.4.1 Regressão Linear

O método de regressão linear é uma forma simples de performar aprendizado supervisionado, ainda assim é um ótimo ponto de inicio para servir de *benchmarking* para modelos mais complicados.

Em suma esse modelo encontra um vetor de pesos W e um bias b que melhor explica uma relação linear entre a entrada x e a saída y, conforme a equação 11.

$$\hat{y} = x \cdot W^T + b \tag{11}$$

Para estimar o vetor de pesos e o termo de bias é formulado um problema de otimização que minimiza a soma dos quadrados do erro residual, conforme equação 12, utilizando o método de mínimos quadrados (JAMES et al., 2013).

$$\arg\min_{W,b} \sum_{i=1}^{n} (y_i - x_i \cdot W^T - b)^2 \tag{12}$$

4.4.2 *Ridge Regression*

Ridge Regression é um modelo de aprendizado muito parecido com a regressão linear, a única diferença é que existe um termo adicional na função objetivo para minimizar a norma dos pesos $w_j \in W$, conforme mostra a formulação 13 (JAMES et al., 2013) .

$$\arg\min_{W,b} \sum_{i=1}^{n} (y_i - x_i \cdot W^T - b)^2 + \lambda \sum_{j=1}^{m} {w_j}^2 \tag{13}$$

O termo λ da equção, chamado de termode penalização, não é otimizado pelo método, mas sim, é um parâmetro escolhido pelo projetista. Esse termo é sempre, $\lambda \geq 0$, e quanto maior mais as variáveis não importantes são próximas de 0 (van Wieringen, 2015). Um caso especial desse termo é $\lambda = 0$, em que, nessa situação a *ridge regression* se torna exatamente igual a uma regressão linear.

Dessa forma, o método de *ridge regression*, têm uma tendência de generalizar melhor o aprendizado em situações em que existem muitas variáveis de entradas que não adicionam informações ao modelo (van Wieringen, 2015).

4.4.3 *Multilayer Perceptron*

Na seção 4.3.3 foi definida a topologia da rede neural artificial MLP para o problema de classificação, entretanto, esse método também pode ser utilizado para problemas de regressão.

Nessa situação as camadas de entrada e intermediária permanecem exatamente iguais, a única mudanção nescessária é na camda de saída. Para regressão a camada de saída é um único neurônio linear conforme mostrado pela equação 14 (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$\phi(z) = z \tag{14}$$

Capítulo 5

Estatísticas de teste

5.1 Definição de estatística de teste

Uma estatística de teste é o valor que é calculado a partir de dados durante um teste de hipóteses. O valor dessa estatística mede o grau de concordância entre uma amostra de dado e a hipótese nula, que por sua vez pode ser rejeitada ou não (CASELLA; BERGER, 2002).

Em um teste estatístico, as hipóteses são premissas a serem testadas. Tradicionalmente existem duas hipóteses, a nula e a alternativa, em que o objetivo do teste é conservadoramente rejeitar a hipótese nula à favor da hipótese alternativa. Dessa forma, os testes são feitos tal que o objetivo que deseja-se testar é descrito pela hipótese alternativa.

5.2 Testes paramétrico e não-paramétrico

Um teste estatístico paramétrico é aquele que faz suposições sobre os parâmetros da distribuição geradora das populações que estão sendo testadas. Desta forma, um teste não-paramétrico é aquele que implica a ausência dessas suposições, de forma mais direta é aquele que não supões nada sobre os parâmetros da função de distribuição de probabilidade geradora (LOWRY; RICHARD, 2014).

Na maioria das situações práticas, os teste não-paramétricos são vantajosos quando as populações de teste são muito pequenas, possuem estrutura ordinal, ou são melhor representadas pela mediana. Quase que em todas as demais situações, os testes paramétricos se comportam de forma mais confiável e geram resultados com maior potência estatística (CASELLA; BERGER, 2002).

Além disso, os teste paramétricos não estão limitados pela suposição de que a dispersão amostral é igual nas populações de teste, diferente da maioria dos não-paramétricos, e

tampouco funciona apenas para funções geradoras normais. Dado um tamanho amostral suficientemente grande, qualquer distribuição pode ser aplicado para um teste que assume normalidade (KERNS, 2010).

5.2.1 ANOVA

O teste estatístico análise de variância, comumente conhecido como ANOVA, permite avaliar se existe uma diferença significativa entre médias de populações diferentes, e se os fatores exercem influência em alguma variável dependente (**??**). Nesse teste, a hipótese nula é que as médias populacionais são iguais e a hipótese alternativa é que pleo menos uma é diferente. Além disso, o ANOVA, é chamado de paramétrico pois assume que as amostras são aleatórias e independetes, que as populações seguem distribuição normal e possuem variância iguais.

No caso do teste entre classificadores em bases de dados diferentes, o ANOVA divide a varabilidade total entre variabilidade entre algoritmos, bases de dados e erro residual. Se a variabilidade entre algoritmos é significantemente maior que a variabilidade do erro, é possível rejeitar a hipótese nula afavor da hipótese alteranativa (DEMŠAR, 2006).

O resultado do teste ANOVA permite afirmar se todos o algoritmos perforam igualmente ou não, é importante notar que ele não diz respeito qual é melhor ou pior que os demais. Para auxiliar com essa informação exitem dois testes que podem ser aplicados aposteriori, teste de Tukey e teste de Dunnett. Para comparar todos os métodos entre sí é necessário a utilização do teste de Tukey, o teste de Dunnet por outro lado compara todos os métoso com um base (DEMŠAR, 2006).

5.2.2 Friedman

O teste de Firedman é um equivalente não paramétrico ao teste ANOVA, em que seu resultado é um rank da performace de todos os métodos. A hipóse nula desse teste é que a diferença entre os pares segue uma distribuição simétrica em torno de 0, e a hipótese alternativa é que não segue essa distribuição.

É importante notar que a principal vantagem do teste de Friendman, quando comparado ao teste ANOVA, é que ele não assume premissas quando a distribuição dos resultados, e mais importe ainda é que ele não assume que as variâncias são iguais. Isso é importante porque não nescessariamente os resultados assumem distribuições normais e tampouco os eles possuem mesma variância para bases diferentes (DEMŠAR, 2006).

Capítulo 6

O pacote MLAT

A tarefa de criação de um novo modelo de aprendizado de máquina passa por algumas etapas bem definidas, como, implementação, *benchmarking* e análise de resultados, conforme mostra o diagrama da figura 7.

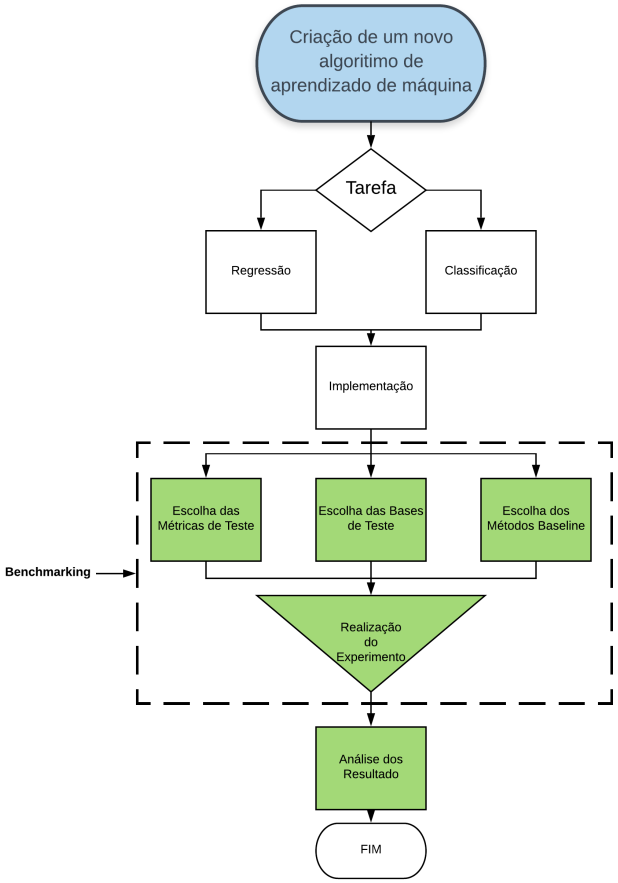


Figura 6 – Etapas da construção de um novo modelo de aprendizado.

O pacote *Machine Learning Atutomated Testing* (MLAT) têm como objetivo automatizar e

simplificar algumas dessas etapas, representadas de verde no diagrama da figura 7.

Nesse capítulo será descrito cada uma dessas etapas e como, o pacote MLAT, vai auxiliar seus usuários em cada uma delas. Por fim, será apresentado a forma de utilização do pacote e uma implementação exemplo.

6.1 Métodos Baseline

Os métodos baseline foram divididos em três grupos : regressão, classificação binário ou *multylabel*. Foi importante dividir os métodos nesses três grupos porque o mesmo foi feito para as bases de dados e métricas de teste. Essa decisão de projeto possibilitou uma simplicidade na interface do usuário, em que ele apenas informa o programa a tarefa e é possível obter todos os métodos, bases e métricas que serão possíveis de serem utilizadas.

No MLAT foram implementados, inicialmente, os métodos descritos no capítulo 4 para as três tarefas citadas acima. O pacote fornece uma interface que o usuário pode apenas informar a tarefa e o programa retorna uma lista com os métodos e suas parametrizações padrão, ou ainda, o usuário pode criar uma lista com os métodos e suas próprias parametrizações.

6.2 Métricas de Testes

Todo algorítimo de aprendizado de máquina, implicitamente ou explicitamente, tem como objetivo minimizar algum tipo de erro. Todavia, o erro minimizado, pode ser diferente entre os modelos, ou ainda, não representar o verdadeiro objetivo do aprendizado. Dessa maneira, durante a comparação entre os modelos de aprendizado, é importante definir a priori métricas representatívas do verdadeiro objetivo do aprendizado, para ser possível comparar os modelos da maneira justa.

Para fazer isso, o pacote MLAT, permite ao usuário escolher entre mais de 20 possíveis métricas de avaliação divididas em três tarefas. Para utilizar qualquer uma das métricas, o usuário, precisa informar ao pacote os nomes das métricas desejadas ou escolher entre as tarefas de regressão, classificação binária ou *multylabel*.

6.3 Bases de Teste

Algumas bases de dados, como MNIST (LECUN, 1998) ou *Spam or Ham* (ALMEIDA; HIDALGO; YAMAKAMI, 2011), se tornaram problemas clássicos para se comparar o desempenho de novos modelos de aprendizado de máquina. Com o objetivo de poupar o usuário da busca e preparo das bases de dados clássicas, foi implementado no pacote uma forma de carregamento em que ele apenas informa os nomes das bases ou a tarefa

de aprendizado. Após a escolha das bases o pacote aplica os modelos em cada uma das bases de maneira autônoma, informando o usuário apenas as métricas de testes.

É importante salientar, que nessa versão do pacote, é possível serem utilizadas apenas as bases de disponibilizadas. Essa foi uma decisão de projeto que garantiu um funcionamento mais otimizado do pacote e que possibilitou uma interface mais simples para o usuário.

Até o momento foram disponibilizadas 20 bases, e na próxima versão do MLAT, haverá uma interface que possibilitará a adição de qualquer base por meio de uma função que à colocará no padrão do pacote.

6.4 Realização do Experimento

A realização do experimento culmina na combinação das três etapas anteriores, descritas nas seções 6.1, 6.2 e 6.3. Nesse momento o usuário pode informar ao programa alguns parâmetros do teste, como o percentual do *split* entre treino e teste, e o número de vezes que cada teste será realizado para os pares de algoritmo e parametrização. Ao fim do processo é entregue ao usuário uma tabela que contêm todos as métricas dos resultados de cada iteração de teste.

6.5 Análise dos Resultados

A etapa de análise dos resultados é o momento em que o projetista deve escolher uma métrica objetivo para que ele possa comparar os diferentes métodos treinados. A saida desse momento deve ser capaz de auxiliar o projista no julgamento de qual método tever melhor desempenho no experimento. Para cumprir esse objetivo, o pacote permite ao usuário escolher e performar um dos testes estatísticos descritos no capítulo 5.

6.6 Uso do MLAT

```
### Carrega e instala pacotes
install.packages('devtools')
require('devtools')
devtools::install_github('PauloCirino/MLAT')
require('MLAT')

### O novo algoritmo
MyNewAlgoFunc <- function(X_train, Y_train, X_test, k) {
Y_hat <- numeric()
```

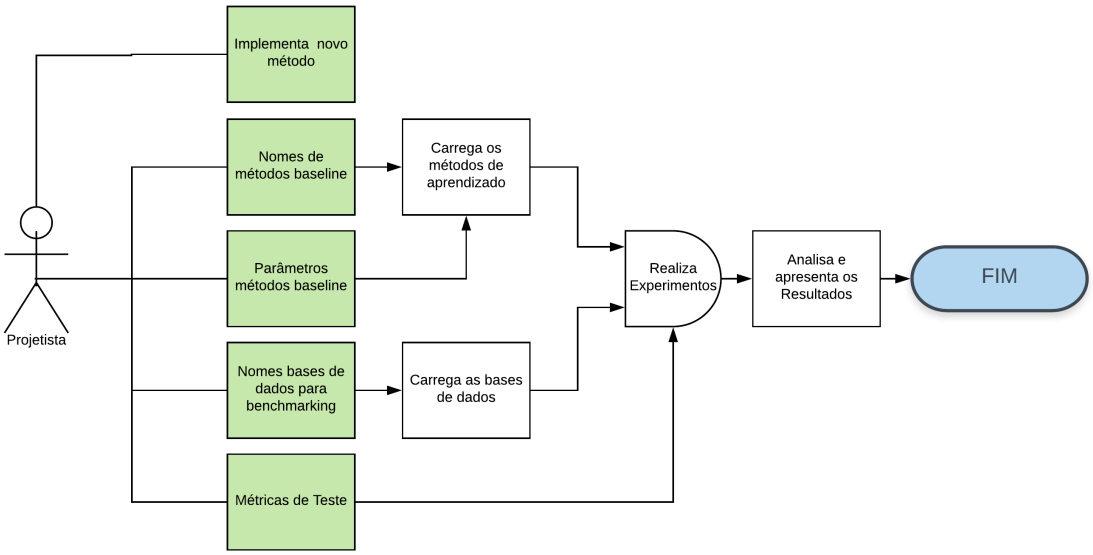


Figura 7 – Utilização do Pacote MLAT.

```
for( i in 1:nrow(X_test)){
x_iter <- X_test[i, ]
distVet <- apply(X_train, 1, function(x){
sum( abs(x_iter - x ) )
})
Y_iter_vet <- Y_train[ order(distVet) ] [1:k]
Y_iter_aux <- numeric()
for(j in 1:k){
Y_iter_aux <- append(Y_iter_aux,
rep(Y_iter_vet[j],
k - j + 1) )
}
unique_y <- unique(Y_iter_aux)
aux_Table <- tabulate(match(Y_iter_aux, unique_y))
Y_hat[i] <- unique_y[which.max(aux_Table)]
}
Y_hat
}

### Parametriza teste
task <- 'MultClass'
cmpTestsFuncsList <- MLAT::GetAllMultClassAlgo()
dataSetNames <- MLAT::GetDataSetsNames(task = task)
```

```
testMetrics <- MLAT::GetMetrics(task = task)

### Coloca minha funcao no padrao do pacote
newAlgoInStandarts <- MLAT::CreateAlgo(
algoName = 'KNN Mahalanobis Ponderado',
algoFun = MyNewAlgoFunc,
task = task,
paramList = list(k = 2:10) )
cmpTestsFuncsList[[length(cmpTestsFuncsList) + 1]] <- newAlgoInStandarts

### Realiza experimento
myResult <- RunTests( cmpTestsFuncsList = cmpTestsFuncsList,
task = task,
dataSetNames = dataSetNames,
metrics = testMetrics,
nTestsPerParam = 10,
splitPerc = 0.7,
verbose = TRUE)
```

REFERÊNCIAS

ABADI, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **CoRR**, abs/1603.04467, 2016. Disponível em: <http://arxiv.org/abs/1603.04467>. Citado 2 vezes nas páginas 7 e 17.

ALMEIDA, T. A.; HIDALGO, J. M. G.; YAMAKAMI, A. **Contributions to the study of SMS spam filtering: new collection and results**. [S.l.]: Proceedings of the 11th ACM symposium on Document engineering, 2011. 259–262 p. Citado na página 24.

BOSER, B. E.; GUYON, I. M.; VAPNIK, V. N. A training algorithm for optimal margin classifiers. In: ACM. **Proceedings of the fifth annual workshop on Computational learning theory**. [S.l.], 1992. p. 144–152. Citado 2 vezes nas páginas 16 e 17.

BRAGA A. P., e. C. A. C. P. L. F. e. L. T. B. **Redes neurais artificiais: teoria e aplicações**. [S.l.]: LTC Editora, 2007. Citado 2 vezes nas páginas 17 e 18.

CASELLA, G.; BERGER, R. L. **Statistical inference**. [S.l.]: Thomson Learning, 2002. 660 p. ISBN 0534243126. Citado na página 21.

CASS, S. The 2017 Top Programming Languages. **IEEE Spectrum**, 2017. Disponível em: <https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages>. Citado na página 8.

CHERKASSKY, V. S.; MULIER, F. **Learning from data : concepts, theory, and methods**. [S.l.]: IEEE Press, 2007. 538 p. ISBN 0471681822. Citado 2 vezes nas páginas 12 e 14.

CORTES, C.; VAPNIK, V. Support-vector networks. **Machine Learning**, Kluwer Academic Publishers, v. 20, n. 3, p. 273–297, sep 1995. ISSN 0885-6125. Disponível em: <http://link.springer.com/10.1007/BF00994018>. Citado na página 16.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. **Journal of Machine learning research**, v. 7, n. Jan, p. 1–30, 2006. Citado na página 22.

GNU Operating System. **Proprietary Software Is Often Malware**. 2017. Disponível em: <https://www.gnu.org/proprietary/proprietary.html>. Citado na página 5.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado 4 vezes nas páginas 12, 17, 19 e 20.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning**. New York, NY: Springer New York, 2009. (Springer Series in Statistics). ISBN 978-0-387-84857-0. Disponível em: <http://link.springer.com/10.1007/978-0-387-84858-7>. Citado 2 vezes nas páginas 14 e 15.

INC, F. S. F. **What is free software?** 2012. Disponível em: <http://www.gnu.org/philosophy/free-sw.html>. Citado na página 3.

JAMES, G. et al. **An Introduction to Statistical Learning**. New York, NY: Springer New York, 2013. v. 103. (Springer Texts in Statistics, v. 103). ISBN 978-1-4614-7137-0. Disponível em: <http://link.springer.com/10.1007/978-1-4614-7138-7>. Citado 4 vezes nas páginas 15, 16, 17 e 19.

KERNS, G. J. **Introduction to probability and statistics using R**. [S.l.]: [s.n.], 2010. ISBN 0557249791. Citado na página 22.

LECUN, Y. **The MNIST database of handwritten digits**. [S.l.: s.n.], 1998. Citado na página 24.

LESSIG, L.; STALLMAN, R. **Free Software, Free Society: Selected Essays of Richard M. Stallman**. [s.n.], 2002. Disponível em: <https://www.gnu.org/philosophy/fsfs/rms-essays.pdf>. Citado 3 vezes nas páginas 3, 6 e 7.

LOWRY, R.; RICHARD, R. Concepts and Applications of Inferential Statistics. **http://vassarstats.net/textbook/**, 2014. Disponível em: <http://doer.col.org/handle/123456789/4853>. Citado na página 21.

NETO, P. C. R. **MLAT: Machine Learning Automated Software**. [S.l.], 2018. R package version 0.1.0. Disponível em: <https://github.com/PauloCirino/MLAT>. Citado na página 15.

OPENHUB. **The R-project (GNU S) Open Source Project on Open Hub**. 2017. <https://www.openhub.net/p/r_project>. (Accessed on 05/30/2018). Citado na página 9.

R-PROJECT. **R: What is R?** 2017. Disponível em: <https://www.r-project.org/about.html>. Citado na página 9.

Richard Stallman. **Why Schools Should Exclusively Use Free Software**. 2017. Disponível em: <https://www.gnu.org/education/edu-schools.en.html>. Citado na página 6.

RILEY, M. **Programming your home : automate with Arduino, Android, and your computer**. [S.l.]: Pragmatic Bookshelf, 2012. 216 p. ISBN 1934356905. Citado na página 6.

SZEGEDY, C. et al. Going deeper with convolutions. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 1–9. Citado na página 7.

TATE, J. et al. **Edward Snowden says motive behind leaks was to expose 'surveillance state'**. 2013. Disponível em: <https://www.washingtonpost.com/politics/edward-snowden-says-motive-behind-leaks-was-to-expose-surveillance-state/2013/06/09/aa3f0804-d13b-11e2-a73e-826d299ff459{_}story.html?tid=pm{_}politics{_}pop{&}utm{_}ter>. Citado na página 5.

TAY, B.; HYUN, J. K.; OH, S. A machine learning approach for specification of spinal cord injuries using fractional anisotropy values obtained from diffusion tensor images. v. 2014, p. 276589, 01 2014. Citado na página 15.

The R Foundation for Statistical Computing. **R: Software Development Life Cycle A Description of R's Development, Testing, Release and Maintenance Processes**. [s.n.], 2014. Disponível em: <https://www.r-project.org/doc/R-SDLC.pdf>. Citado 2 vezes nas páginas 9 e 10.

Top 500 project. **Operating system Family - Systems share**. 2017. Disponível em: <https://www.top500.org/lists/2017/11/>. Citado na página 6.

TOZZI, C.; ZITTRAIN, J. **For Fun and Profit: A History of the Free and Open Source Software Revolution**. MIT Press, 2017. (History of Computing). ISBN 9780262036474. Disponível em: <https://books.google.com.br/books?id=MXosDwAAQBAJ>. Citado na página 4.

van Wieringen, W. N. Lecture notes on ridge regression. **ArXiv e-prints**, set. 2015. Citado 2 vezes nas páginas 19 e 20.

VAPNIK, V. N.; CHERVONENKIS, A. Y. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. **Theory of Probability & Its Applications**, Society for Industrial and Applied Mathematics, v. 16, n. 2, p. 264–280, jan 1971. ISSN 0040-585X. Disponível em: <http://epubs.siam.org/doi/10.1137/1116025>. Citado na página 14.

W3TECHS. **Usage of operating systems for websites**. 2017. Disponível em: <https://w3techs.com/technologies/overview/operating{_}system/>. Citado na página 6.

WICKHAM, H. **Advanced R**. CRC Press, 2015. Disponível em: <https://englianhu.files.wordpress.com/2016/05/advanced-r.pdf>. Citado 2 vezes nas páginas 8 e 9.

WILLIAMS, S.; STALLMAN, R. M. Free as in Freedom (2.0): Richard Stallman and the Free Software Revolution Second edition revisions. 2010. Disponível em: <https://sagitter.fedorapeople.org/faif-2.0.pdf>. Citado 3 vezes nas páginas 3, 4 e 5.

ZAMBIASI, S. P.; RABELO, R. J. Uma arquitetura aberta e orientada a serviços para softwares assistentes pessoais. **RITA**, v. 19, p. 93–119, 2012. Citado na página 1.

Apêndices

APÊNDICE A – MANUAL DO PACOTE MLAT

Package 'MLAT'

May 30, 2018

Type Package

Title Machine Learning Automated Software

Date 2018-05-29

Version 0.1.0

Author Paulo Cirino Ribeiro Neto

Maintainer Paulo Cirino Ribeiro Neto <paulocirino.neto@gmail.com>

Description The package was created to help R users automate the processes
of benchmarking machine learning methods agains common methods in common datasets.

License GPL-3 + file LICENSE

Encoding UTF-8

LazyData false

RoxygenNote 6.0.1

Imports e1071,
class

Suggests ggplot2

R topics documented:

`Accuracy` *Accuracy*

Description

Returns the Accuracy for a classification problem.

Usage

```
Accuracy(Y, Y_hat)
```

Arguments

`Y`	Ground truth numeric vector.
`Y_hat`	Predicted Labels numeric vector.

Value

A numeric value corresponding to the Accuracy of a classification problem

Examples

```
Y = sample(x = c(1,2), size = 10, replace = TRUE)
Y_hat = sample(x = c(1,2), size = 10, replace = TRUE)
Accuracy(Y = Y, Y_hat = Y_hat)
```

`AUC` *AUC*

Description

Returns the Area Under the Curve for a binarry classification problem.

Usage

```
AUC(Y, Y_hat)
```

Arguments

`Y`	Ground truth numeric vector.
`Y_hat`	Predicted Labels numeric vector.

Value

A numeric value corresponding to the AUC of binary classification problem

Examples

```
Y = sample(x = c(1,2), size = 10, replace = TRUE)
Y_hat = sample(x = c(1,2), size = 10, replace = TRUE)
AUC(Y = Y, Y_hat = Y_hat)
```

BinClassMetrics *BinClassMetricsNames*

Description

Returns a binaryResultList with binary classificaiton metrics.

Usage

```
BinClassMetrics(Y, Y_hat, MetricsNames = BinClassMetricsNames())
```

Arguments

Y	Ground truth numeric vector.
Y_hat	Predicted Labels numeric vector.
MetricsNames	can be found at BinClassMetricsNames()

Value

A binaryResultList with results

Examples

```
Y = sample(x = c(1,2), size = 10, replace = TRUE)
Y_hat = sample(x = c(1,2), size = 10, replace = TRUE)
BinClassMetrics(Y = Y, Y_hat = Y_hat, MetricsNames = BinClassMetricsNames())
```

BinClassMetricsNames *Binary Classification Metrics*

Description

Returns a character string vector containning all binary classification metrics.

Usage

```
BinClassMetricsNames()
```

Value

character string vector with all possible binary classification metrics.

Examples

```
BinClassMetricsNames()
```

CreateAlgo *Create Algorithm*

Description

It is an auxiliary function to help creating new algorithms in the package standarts.

Usage

```
CreateAlgo(algoName, algoFun, task, paramList)
```

Arguments

algoName	A character string that represents the algorithm name.
algoFun	A function class object.
task	A character string vector, cointaning 'MultClass' or/and 'BinClass' and/or 'Regression'.
paramList	A list of all parameters and their values to be tested.

Value

An object of class algorithm.

Examples

```
algoName <- 'myAlgo'
algoFun <- function(X_train, Y_train, X_test, param1){
 set.seed(param1)
 sample(Y_train, size = nrow(X_test), replace = TRUE)}
task <- c('MultClass', 'BinClass')
paramList = list(param1 = 1:3)
CreateAlgo(algoName = algoName, algoFun = algoFun, task = task, paramList = paramList)
```

CreateDataSet *Create DataSet*

Description

Creates a DataSet object type, to be used with the models provided with this package

Usage

```
CreateDataSet(X, Y, Name, type, task)
```

Arguments

X	A Matrix.
Y	A numeric vector of classes or values.
Name	A character string, as dataset name.
type	A character string, the types of values for X (numeric or integer).
task	A character string vector of task to be performed, check GetPossibleTasks().

Value

A DataSet object type.

Examples

```
X <- as.matrix(cbind(runif(n = 100), runif(n = 100)))
Y <- sample(x = c(1, 2), size = 100, replace = TRUE)
Name <- 'randomData'
type <- 'numeric'
task <- 'BinClas'
newData <- CreateDataSet(X = X, Y = Y, Name = Name, type = type, task = task)
```

GetAllMultClassAlgo *Generates all MultClass Classification Alogrithms*

Description

It is an auxiliary that allows to load all of the packages multclass classification alogrithms.

Usage

```
GetAllMultClassAlgo()
```

Value

Returns a list with all MultClass Classification Alogrithms

Examples

```
GetAllMultClassAlgo()
```

GetData *Get DataSet by name*

Description

Get a dataset by name

Usage

```
GetData(datasetName, seed, splitPerc)
```

Arguments

datasetName	A character string, as DataSet name.
seed	For the traint and test split.
splitPerc	Percentage for train of all dataSet, between 0 and 1.

Value

A trainTestDataSet object type.

Examples

```
GetData(datasetName = 'Iris', seed = 123, splitPerc = 0.7)
```

GetDataSetsNames *Get Datasets Names*

Description

Get all available datasets names

Usage

```
GetDataSetsNames(task)
```

Arguments

task	A character string with desired task, for possible tasks check GetPossibleTasks().

Value

A character string vector with all possible datasets names.

Examples

```
GetDataSetsNames()
```

GetMetrics *Get Metrics*

Description

Get all possible task names

Usage

```
GetMetrics(task)
```

Arguments

A	valid task, can be 'MultClass' or 'BinClass'.

Value

All metrics names for that taks

Examples

```
GetPossibleTasks()
```

`GetPossibleTasks` *Get Possible Tasks*

Description

Get all possible task names

Usage

```
GetPossibleTasks()
```

Value

A character string vector with all possible task names.

Examples

```
GetPossibleTasks()
```

`knn` *K Nearest Neighbours*

Description

It is the K Nearest Neighbours method

Usage

```
knn(X_train, Y_train, X_test, K)
```

Arguments

`X_train`	A Matrix of trainning observations.
`Y_train`	A numeric vector of classes or values of the trainning observations.
`X_test`	A Matrix of testing observations.
`K`	An integer as a parameter for the knn method.

Value

predicted labels

Examples

```
X <- as.matrix(cbind(runif(n = 100), runif(n = 100)))
pos <- sample(100, 70)
X_train <- X[pos, ]
X_test <- X[-pos, ]
Y_train <- as.numeric( X_train[, 1] ** 2 - X_train[, 2] > 0)
Y_test <- as.numeric(X_test[, 1] ** 2 - X_test[, 2] > 0)
K <- 5
Y_predicted <- knn(X_train = X_train, Y_train = Y_train, X_test = X_test, K = K)
print(table(Y_test, Y_predicted))
```

LogLoss *LogLoss*

Description

Returns the Logarithmic Loss for classification problem.

Usage

```
LogLoss(Y, Y_hat)
```

Arguments

Y	Ground truth numeric vector.
Y_hat	Predicted Labels numeric vector.

Value

A numeric value corresponding to the LogLoss of binary classification problem

Examples

```
Y = sample(x = c(1,2), size = 10, replace = TRUE)
Y_hat = sample(x = c(1,2), size = 10, replace = TRUE)
LogLoss(Y = Y, Y_hat = Y_hat)
```

MultClassMetrics *MultClassMetrics*

Description

Returns a multiClassResultList with multi class classificaiton metrics.

Usage

```
MultClassMetrics(Y, Y_hat, MetricsNames)
```

Arguments

Y	Ground truth numeric vector.
Y_hat	Predicted Labels numeric vector.
MetricsNames	Metrics names, avilable can be found with MultiClassMetricsNames() .

Value

A multiClassResultList with results

Examples

```
Y = sample(x = c(1,2, 3), size = 20, replace = TRUE)
Y_hat = sample(x = c(1, 2, 3), size = 20, replace = TRUE)
MultClassMetrics(Y = Y, Y_hat = Y_hat, MetricsNames = MultiClassMetricsNames())
```

MultiClassMetricsNames *Multi Class Classification Metrics*

Description

Returns a character string vector containning all multi class classification metrics.

Usage

```
MultiClassMetricsNames()
```

Value

character string vector with all possible multi class classification metrics.

Examples

```
MultiClassMetricsNames()
```

MultiLogLoss *MultiLogLoss*

Description

Returns the Logarithmic Loss for multi class classification problem.

Usage

```
MultiLogLoss(Y, Y_hat)
```

Arguments

Y	Ground truth numeric vector.
Y_hat	Predicted Labels numeric vector.

Value

A numeric value corresponding to the LogLoss of binary classification problem

Examples

```
Y = sample(x = c(1,2), size = 10, replace = TRUE)
Y_hat = sample(x = c(1,2), size = 10, replace = TRUE)
MultiLogLoss(Y = Y, Y_hat = Y_hat)
```

RunTests *Run Tests*

Description

Runs the tests given the methods and datasets

Usage

```
RunTests(cmpTestsFuncsList = cmpTestsFuncsList, task = task,
  dataSetNames = dataSetNames, metrics = NA, nTestsPerParam = 1,
  splitPerc = 0.7, verbose = TRUE)
```

Arguments

cmpTestsFuncsList	is a list of
task	A character string with 'MultClass' or 'BinClas'
dataSetNames	A character string vector with valid dataset names, for options check
metrics	character string vector with all testing metrics or NA for all available metrics
nTestsPerParam	FOOO
splitPerc	TODO
verbose	TRUE/FALSE value for printing partial test

Value

TODO

Examples

```
cmpTestsFuncsList <- GetAllMultClassAlgo()
task <- 'MultClass'
dataSetNames <- c('Iris', 'PimaIndiansDiabetes')
myResult <- RunTests(cmpTestsFuncsList = GetAllMultClassAlgo(),
                     task = 'MultClass',
                     dataSetNames = c('Iris', 'PimaIndiansDiabetes'))
```

squareConfusionTable *Square Confusion Matrix*

Description

Returns a square confusion matrix

Usage

```
squareConfusionTable(Y, Y_hat)
```

Arguments

Y	A numeric vector for the ground truth labels
Y_hat	A numeric vector for the predicted Labels

Value

A confusion table

Examples

```
squareConfusionTable(Y = sample(1:2, size = 10, replace = TRUE), Y_hat = rep(1, 10))
```

---

svm_linear *Linear Support Vector Machines*

Description

It is the Support Vector Machines without a kernel

Usage

```
svm_linear(X_train, Y_train, X_test, C)
```

Arguments

X_train	A Matrix of trainning observations.
Y_train	A numeric vector of classes or values of the trainning observations.
X_test	A Matrix of testing observations.
C	A numeric value that represents the cost of constraints violation of the regularization term in the Lagrange formulation.

Value

predicted labels

Examples

```
X <- as.matrix(cbind(runif(n = 100), runif(n = 100)))
pos <- sample(100, 70)
X_train <- X[pos, ]
X_test <- X[-pos, ]
Y_train <- as.numeric( X_train[, 1] ** 2 - X_train[, 2] > 0)
Y_test <- as.numeric(X_test[, 1] ** 2 - X_test[, 2] > 0)
C <- 5
Y_predicted <- svm_linear(X_train = X_train, Y_train = Y_train, X_test = X_test, C = C)
print(table(Y_test, Y_predicted))
```

svm_poli *Support Vector Machines with Polinomial Kernel*

Description

It is the Support Vector Machines with a polinomial kernel

Usage

```
svm_poli(X_train, Y_train, X_test, degree, gamma, coef0, C)
```

Arguments

X_train	A Matrix of trainning observations.
Y_train	A numeric vector of classes or values of the trainning observations.
X_test	A Matrix of testing observations.
degree	A integer that represents the kernel polynomial degree
gamma	A numeric value as the kernel coefficient.
coef0	A numeric value for kernel independent term.
C	A numeric value that represents the cost of constraints violation of the regularization term in the Lagrange formulation.

Value

predicted labels

Examples

```
X <- as.matrix(cbind(runif(n = 100), runif(n = 100)))
pos <- sample(100, 70)
X_train <- X[pos, ]
X_test <- X[-pos, ]
Y_train <- as.numeric( X_train[, 1] ** 2 - X_train[, 2] > 0)
Y_test <- as.numeric(X_test[, 1] ** 2 - X_test[, 2] > 0)
C <- 5
coef0 <- 0
degree <- 5
gamma <- 0.5
Y_predicted <- svm_poli(X_train = X_train, Y_train = Y_train, X_test = X_test, C = C, coef0 = coef0, degree = deg
print(table(Y_test, Y_predicted))
```

svm_radial *Support Vector Machines with Radial Kernel*

Description

It is the Support Vector Machines with a radial kernel

Usage

```
svm_radial(X_train, Y_train, X_test, gamma, C)
```

Arguments

X_train	A Matrix of trainning observations.
Y_train	A numeric vector of classes or values of the trainning observations.
X_test	A Matrix of testing observations.
gamma	A numeric value as the kernel coefficient.
C	A numeric value that represents the cost of constraints violation of the regularization term in the Lagrange formulation.

Value

predicted labels

Examples

```
X <- as.matrix(cbind(runif(n = 100), runif(n = 100)))
pos <- sample(100, 70)
X_train <- X[pos, ]
X_test <- X[-pos, ]
Y_train <- as.numeric( X_train[, 1] ** 2 - X_train[, 2] > 0)
Y_test <- as.numeric(X_test[, 1] ** 2 - X_test[, 2] > 0)
C <- 5
gamma <- 0.5
Y_predicted <- svm_radial(X_train = X_train, Y_train = Y_train, X_test = X_test, C = C, gamma = gamma)
print(table(Y_test, Y_predicted))
```

Index