Pontifícia Universidade Católica
do Rio de Janeiro

Felipe Reis Gomes

Classificação de ofertas de produtos

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC-Rio

Orientador : Prof. Marco Antonio Casanova
Co–Orientador: Prof. Ruy Luiz Milidiú

Rio de Janeiro
Dezembro de 2012

PUC-Rio - Certificação Digital Nº 1012636/CA

Pontifícia Universidade Católica
do Rio de Janeiro

Felipe Reis Gomes

Classificação de ofertas de produtos

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Marco Antonio Casanova
Orientador
Departamento de Informática — PUC-Rio

Prof. Ruy Luiz Milidiú
Co–Orientador
Departamento de Informática — PUC-Rio

Prof. Antônio Luz Furtado
Departamento de Informática — PUC-Rio

Prof. Helio Cortes Vieira Lopes
Departamento de Informática — PUC-Rio

Prof. José Eugenio Leal
Coordenador Setorial do Centro
Técnico Científico — PUC-Rio

Rio de Janeiro, 18 de Dezembro de 2012

PUC-Rio - Certificação Digital Nº 1012636/CA

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador.

Felipe Reis Gomes

Graduou-se no curso de Bacharelado em Informática pela Pontifícia Universidade Católica do Rio de Janeiro. Trabalhou desenvolvendo sistemas de localização geográfica de veículos no TecGraf. Trabalhou na GAPSO - empresa da área de otimização de logística construindo um *framework* de desenvolvimento seguindo as melhores práticas do mercado e na RiskControl - empresa do mercado financeiro. Atual líder técnico do departamento de Desenvolvimento de Sistemas da FINEP (Financiadora de Estudos e Projetos), onde é responsável por definir os padrões de trabalho e as tecnologias que serão utilizadas pelo departamento e pelos fornecedores da empresa.

PUC-Rio - Certificação Digital Nº 1012636/CA

Ficha Catalográfica

Gomes, Felipe Reis

Classificação de ofertas de produtos / Felipe Reis Gomes; orientador: Marco Antonio Casanova; co–orientador: Ruy Luiz Milidiú. — Rio de Janeiro : PUC-Rio, Departamento de Informática, 2012.

v., 84 f: il. ; 29,7 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Tese. 2. Aprendizado de Máquina. 3. Aprendizado Supervisionado. 4. Classificação de Entidades. 5. Classificação de Produtos. 6. WEKA. 7. Framework Aprendizado de Máquina. I. Casanova, Marco Antonio. II. Milidiú, Ruy Luiz. III. Furtado, Antônio Luz. IV. Lopes, Helio Cortes Vieira. V. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. VI. Título.

CDD: 004

Agradecimentos

PUC-Rio - Certificação Digital Nº 1012636/CA

A Deus por me guiar durante toda a minha vida.

Aos meus orientadores Marco Antonio Casanova e Ruy Luiz Milidiú, por terem partilhado comigo dos seus valorosos ensinamentos.

À PUC-Rio e ao TecGraf pelos auxílios concedidos, sem os quais não seria possível realizar este trabalho.

À minha esposa e aos meus familiares, que me apoiaram durante todo o tempo e souberam compreender as minhas ausências ao longo desses anos.

Aos meus amigos que me deram força para transpor os muitos obstáculos encontrados pelo caminho.

Resumo

Gomes, Felipe Reis; Casanova, Marco Antonio; Milidiú, Ruy Luiz. **Classificação de ofertas de produtos**. Rio de Janeiro, 2012. 84p. Dissertação de Mestrado — o de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho apresenta o *EasyLearn*, um *framework* para apoiar o desenvolvimento de aplicações voltadas ao aprendizado supervisionado. O *EasyLearn* define uma camada intermediaria, de simples configuração e entendimento, entre a aplicação e o WEKA, um *framework* de aprendizado de máquina criado pela Universidade de Waikato. Todos os classificadores e filtros implementados pelo WEKA podem ser facilmente encapsulados para serem utilizados pelo *EasyLearn*. O *EasyLearn* recebe como entrada um conjunto de arquivos de configuração no formato XML contendo a definição do fluxo de processamento a ser executado, além da fonte de dados a ser processada, independente do formato. Sua saída é adaptável e pode ser configurada para produzir, por exemplo, relatórios de acurácia da classificação, a própria da fonte de dados classificada, ou o modelo de classificação já treinado. A arquitetura do *EasyLearn* foi definida após a análise detalhada dos processos de classificação, permitindo identificar inúmeras atividades em comum entre os três processos estudados (aprendizado, avaliação e classificação). Através desta percepção e tomando as linguagens orientadas a objetos como inspiração, foi criado um *framework* capaz de comportar os processos de classificação e suas possíveis variações, além de permitir o reaproveitamento das configurações, através da implementação de herança e polimorfismo para os seus arquivos de configuração. A dissertação ilustra o uso do *framework* criado através de um estudo de caso completo sobre classificação de produtos do comércio eletrônico, incluindo a criação do corpus, engenharia de atributos e análise dos resultados obtidos.

Palavras–chave

Aprendizado de Máquina; Aprendizado Supervisionado; Classificação de Entidades; Classificação de Produtos; WEKA; Framework Aprendizado de Máquina;

PUC-Rio - Certificação Digital Nº 1012636/CA

Abstract

Gomes, Felipe Reis; Casanova, Marco Antonio; Milidiú, Ruy Luiz. **Product Offering Classification**. Rio de Janeiro, 2012. 84p. MSc Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This dissertation presents EasyLearn, a framework to support the development of supervised learning applications. EasyLearn defines an intermediate layer, which is easy to configure and understand, between the application and WEKA, a machine learning framework created by the University of Waikato. All classifiers and filters implemented by WEKA can be easily encapsulated to be used by EasyLearn. EasyLearn receives as input a set of configuration files in XML format containing the definition of the processing flow to be executed, in addition to the data source to be classified, regardless of format. Its output is customizable and can be configured to produce classification accuracy reports, the classified data source, or the trained classification model. The architecture of EasyLearn was defined after a detailed analysis of the classification process, which identified a set of common activities among the three analyzed processes (learning, evaluation and classification). Through this insight and taking the object-oriented languages as inspiration, a framework was created which is able to support the classification processes and its variations, and which also allows reusing settings by implementing inheritance and polymorphism in their configuration files. This dissertation also illustrates the use of the created framework presenting a full case study about e-commerce product classification, including corpus creation, attribute engineering and result analysis.

Keywords

Machine Learning; Supervised Learning; Entity Classification; Product Classification; WEKA; Machine Learning Framework;

PUC-Rio - Certificação Digital Nº 1012636/CA

Sumário

PUC-Rio - Certificação Digital Nº 1012636/CA

PUC-Rio - Certificação Digital Nº 1012636/CA

Lista de figuras

PUC-Rio - Certificação Digital Nº 1012636/CA

Lista de tabelas

PUC-Rio - Certificação Digital Nº 1012636/CA

1
Introdução

Após a revolução tecnológica proporcionada pela popularização da internet na década de 1990, inovadores modelos de negócio surgiram para rivalizar com as formas tradicionais de trabalho. Uma das evoluções mais notáveis trazidas pela grande rede foi o *ecommerce* que permitiu encurtar o caminho entre vendedores e compradores e tornou mais fácil o acesso à bens e serviços, independente das fronteiras entre os países.

Segundo um balanço semestral sobre a expansão do comércio eletrônico no Brasil emitido pela consultoria eBit, até o final do primeiro semestre de 2011 mais de 4000 mil lojas virtuais conquistaram o selo de confiança da empresa correspondendo a um aumento de 26% em relação ao mesmo período de 2010 (WebShoppers11). No país são mais de 23.000 lojas virtuais (eSchool11) comercializando milhões de produtos, juntas faturaram em 2010 mais de 14,5 bilhões de reais.

Infelizmente é difícil determinar a quantidade exata de SKU (Stock Keeping Units), que é terminologia usada para identificar de forma única as classes dos produtos comercializados (Tompkins98), pois não existe nenhum diretório unificado de lojas. No entanto, é possível ter uma noção da ordem de grandeza para a contagem de produtos quando observamos os catálogos das três maiores lojas virtuais da internet brasileira (Americanas, Submarino e Saraiva) que possuem mais de um milhão de produtos cadastrados.

Manter catálogos desta magnitude não é uma tarefa simples e agrega diversas subtarefas, como categorização dos produtos, busca por relevância, consolidação de catálogos distintos, detecção de duplicatas, entre outras.

O problema de *classificação de produtos* abordado neste trabalho pode ser definido da seguinte forma: dada uma série de descrições de ofertas de produtos, precisamos identificar quais descrições correspondem ao mesmo produto. Supondo que duas grandes empresas detentoras de web sites de *ecommerce* realizaram uma fusão de seus negócios e agora desejam mesclar seus catálogos. Como minimizar o trabalho manual na identificação dos produtos repetidos entre os catálogos?

Geralmente, as lojas disponibilizam grande parte da descrição e ca-

PUC-Rio - Certificação Digital Nº 1012636/CA

racterísticas de um produto em seu nome de exibição. Assim, por exemplo, "`MONIT LCD 19 ACER X193W WIDE PRETO`" diz respeito a um monitor LCD, de 19 polegadas, da marca ACER, modelo X193W, *widescreen* de cor preta. Esse mesmo produto, em outra loja, tem a descrição "`Monitor LCD Acer ETCX3WP004 X193W ABD 19 20.000:1`". Note que a falta de estrutura na descrição do produto torna a tarefa mais difícil.

Neste trabalho utilizamos técnicas de aprendizado supervisionado para apresentar uma solução ao problema de classificação de ofertas de produtos do comércio eletrônico, cujo o objetivo é identificar se uma determinada oferta de um produto em uma loja virtual corresponde a um produto já catalogado previamente. Para cumprir o objetivo deste trabalho construímos um corpus para apoiar as técnicas de aprendizado supervisionado, realizamos a engenharia de atributos para extrair atributos relevantes a partir da descrição da oferta - que é o único atributo disponível à priori, e por fim, criamos o *EasyLearn*, um *framework* para apoiar o desenvolvimento de aplicações voltadas ao aprendizado supervisionado. Utilizamos o *framework* desenvolvido para resolver o problema supracitado de forma simples e elegante. Além disto, demonstramos o potencial da infra-estrutura criada apresentando outras aplicações relacionadas ao aprendizado supervisionado.

O *EasyLearn* define uma camada intermediaria - de simples configuração e entendimento - entre a aplicação e o WEKA (framework de Aprendizado de Máquina criado pela Universidade de Waikato). Todos os classificadores e filtros implementados pelo *WEKA* podem ser facilmente encapsulados para serem utilizados pelo *EasyLearn*.

O *EasyLearn* recebe como entrada um conjunto de arquivos de configuração no formato XML. A composição destes arquivos irá definir o fluxo de processamento a ser executado, além do DataSource a ser processado - independente do formato. Sua saída é customizável e pode ser configurada para produzir, por exemplo, relatórios de acurácia da classificação, o próprio DataSource classificado, ou o modelo de classificação já treinado.

Os arquivos de configuração são baseados no domínio do problema a ser modelado. No anexo B vemos um exemplo de artefatos de configuração utilizados como entrada do *EasyLearn*.

A arquitetura do *EasyLearn* foi definida após a análise detalhada dos processos de Classificação, onde identificamos inúmeras atividades em comum entre os três processos estudados (aprendizado, avaliação e classificação). Através desta percepção e tomando as linguagens Orientadas à Objetos como inspiração, criamos um *framework* capaz de comportar os processos de classificação e suas possíveis variações, além de permitir o reaproveitamento

PUC-Rio - Certificação Digital Nº 1012636/CA

das configurações, através da implementação de herança e polimorfismo para os seus arquivos de configuração.

Esta dissertação foi organizada da seguinte forma: o capítulo (2) apresenta alguns fundamentos e estratégias de Aprendizado de Máquina, incluindo alguns trabalhos relacionados. O capítulo 3 expõe os processos de classificação utilizados como base da construção do *EasyLearn*. O capítulo 4 fornece uma visão detalhada sobre a arquitetura do *framework*, juntamente com seu escopo e requisitos. O capítulo 5 exemplifica a utilização do *framework* em um estudo de caso real - que é o tema deste trabalho. Finalmente, o capítulo 6 conclui o estudo e relaciona alguns possíveis caminhos para dar continuidade à esta pesquisa.

PUC-Rio - Certificação Digital Nº 1012636/CA

2
Fundamentos

2.1
Classificação

Digamos que queremos reconhecer um número escrito à mão, digitalizado e salvo numa imagem, como visto na figura 2.1. Para cada algarismo, temos 10 possíveis respostas, de 0 à 9. A caligrafia pode variar muito para cada pessoa, e tais variações dificultam a análise. Além do estilo de escrita, existem outros fatores que podem aumentar a complexidade da tarefa: podemos ter imagens de números escritos com lápis, ou caneta, caracteres podem ter tamanhos distintos, etc. No entanto, sabemos que existe um padrão para cada algarismo, senão as pessoas não conseguiriam trocar informações. É este padrão que queremos aprender a identificar neste exemplo (Hastie08).

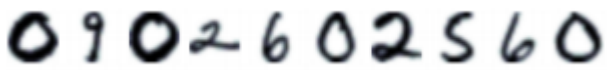

Figura 2.1: Exemplo de escrita manual digitalizada

Este é um problema de OCR (*Optical Character Recognition*) (Alpaydin04) e sua solução pode ser usada, por exemplo, como módulo de reconhecimento de códigos postais, pertencente a sistema de endereçamento automático de correspondências dos Correios. Este desafio, faz parte do repertório de problemas que podem ser resolvidos com Aprendizado de Máquina, mais precisamente, usando as técnicas de classificação, mencionadas nas seções subsequentes. De fato, considere 10 classes, uma para cada algarismo. O objetivo deste problema é identificar, com base em um conjunto de pontos extraído de um pedaço de uma imagem nunca vista até então, o algarismo que melhor representa a parte analisada.

Além do reconhecimento de escrita manual já citado, existem muitas outras aplicações para o Aprendizado de Máquina, incluindo previsão da variação dos preços de ativos nas bolsas de valores, filtro anti-spam, auxílio no diagnóstico de doenças e resultado de exames médicos, detecção de fraudes, re-

PUC-Rio - Certificação Digital Nº 1012636/CA

conhecimento de impressões digitais, da íris, ou de voz (Alpaydin04), predição do risco de crédito (*Credit Scoring*) (Hand98), etc.

2.2 Aprendizado Supervisionado

O problema de classificação, visto como exemplo no início deste capítulo pode ser considerado um problema de Aprendizado Supervisionado, uma vez que temos um domínio de resposta bem definido com 10 possíveis classes (uma para cada algarismo) (Hastie08).

Neste tipo de exemplo, recebemos uma entrada contendo um conjunto de dados históricos - também chamado de corpus, corpora, *dataset*, ou experiência de treinamento (Mitchell97), e para cada instância deste conjunto, temos a classe correta já mapeada. A partir destes dados, treinamos algoritmos de aprendizado supervisionado para aprenderem as regras de associação e os padrões que levam cada instância à sua classe.

Após a etapa de treinamento, existe a fase de validação, que é onde calculamos a taxa de sucesso obtido pelo modelo proposto. O processo de melhora da acurácia é geralmente árduo, pois exige a repetição das etapas de alteração do modelo, treinamento e validação inúmeras vezes, até alcançarmos resultados satisfatórios para a tarefa escolhida.

Explicamos os processos de aprendizado supervisionado com mais detalhes no capítulo 3.

2.3 Aprendizado Não-supervisionado

Enquanto no problema de OCR, elaborado para introduzir os fundamentos básicos do capítulo, temos a quantidade de classes pre-definida, nos problemas de Aprendizado Não-supervisionado, não possuímos à priori o mapeamento entre as instâncias e as classes (Alpaydin04).

Uma forma de organizarmos nosso *dataset* quando não temos as classes bem definidas é utilizando uma técnica de Aprendizado Não-supervisionado, chamada de Clusterização, ou *Clustering*. Neste método, o objetivo é, através da análise dos dados, estabelecer a existência de *clusters*, ou agrupamentos, empregando uma função para calcular a similaridade entre cada instância (Michie94).

Assim como a Classificação é um método de Aprendizado Supervisionado, a Clusterização é um método de Aprendizado Não-Supervisionado.

PUC-Rio - Certificação Digital Nº 1012636/CA

Nosso foco principal não é nos métodos de Aprendizado Não-supervisionado, entretanto, planejamos, futuramente, expandir o trabalho realizado para englobar novos métodos de Aprendizado de Máquina.

2.4 Engenharia de Atributos

Deixemos o problema de OCR de lado, e retomemos o foco para o assunto do estudo de caso abordado no capítulo 5: classificação de produtos.

Segundo (Thor), podemos considerar o problema de classificação de produtos como um problema de particionamento. Dado um conjunto de ofertas de produtos $O = \{o_1, o_2, ..., o_n\}$, o resultado do processo de classificação é o particionamento de O, isto é, um conjunto de partições sem sobreposição $P = \{p_1, p_2, ..., p_k\}$ onde $p_i \cap p_j = \emptyset$, para $1 \leq i \leq j \leq k$ e $p_1 \cup p_2 \cup ... \cup p_k = O$. O objetivo é que todas as ofertas o_i da partição p_j representem o mesmo produto do mundo real e que todas as demais ofertas se referenciem a produtos diferentes.

Para cada instância o_i, temos vinculados alguns atributos, tais como a descrição e a categoria. No entanto, nenhum dos algoritmos utilizados aceitam *texto* como entrada, deste modo, precisamos converter as descrições de ofertas numa estrutura denominada de saco de palavras (*bag of words* (Maron61) e (Lewis98)) e criar uma representação vetorial para cada o_i, como na tabela 2.1.

Esta é a abordagem mais trivial para o problema, e consiste em gerar um dicionário **W** contendo todas as palavras que fazem parte de cada elemento de **X**, o que nos permite atribuir um número único para cada palavra do dicionário - através da sua posição no dicionário. Em seguida, basta contarmos o número de vezes que cada palavra w_j aparece para cada elemento o_i. Para a seguinte entrada, por exemplo:

o_1 = `"MONITOR LCD 19 ACER X193W WIDE PRETO"`
o_2 = `"MONITOR LCD ACER X193W ABD 19 20000:1"`

É criado o dicionário abaixo:
W = `{MONIT,LCD,19,ACER,X193W,WIDE,PRETO,ABD,20000:1}`

Após a construção da representação vetorial, obtemos a matriz a seguir:

Através da representação vetorial das palavras é possível computar distâncias, similaridade e outras estatísticas (Smola08).

PUC-Rio - Certificação Digital Nº 1012636/CA

A técnica descrita acima é apenas um dos artifícios utilizados para realizar a *Engenharia de Atributos*, durante a fase de modelagem da solução proposta. Na seção 3.5 descrevemos outras técnicas disponíveis para pré-processar as instâncias e no capítulo 5 explicamos algumas abordagens empregadas para tratar o problema elaborado no estudo de caso.

2.5 Seleção de Atributos

Conforme a quantidade de atributos cresce - e o modelo vai se tornando mais elaborado, o processo de treino e validação pode se tornar mais demorado para alguns classificadores, já que a complexidade dos algoritmos, eventualmente, está relacionada com o total de atributos analisados. No entanto, nem todos os atributos trazem informações relevantes para a classificação. Identificar os atributos que oferecem maior *Ganho de Informação* (Quinlan93) para a solução e descartar aqueles que reduzem a acurácia faz parte de um processo chamado de *Seleção de Atributos*, e pode ser empregado para melhorar a eficiência da solução.

Existem diversas técnicas para a Seleção de Atributos tais como *Proportional Difference* (Simeon08), *Fisher Kernel* (Jaakkola98), etc.

2.6 Trabalhos Relacionados

Em 1993 a Universidade de Waikato, na Nova Zelândia começou o desenvolvimento do WEKA (Waikato Environment for Knowledge Analysis). O WEKA, além da sua API de desenvolvimento escrita em Java, também possui um conjunto de interfaces gráficas que possibilita sua utilização de uma forma simplificada, mesmo sem nenhum conhecimento em desenvolvimento de aplicações. Por isto, o WEKA foi intitulado como **Ambiente** de Analise de Conhecimento já que desempenha tarefas muito além de um *framework*. Em função da sua facilidade de uso, o WEKA ganhou notoriedade na área de Aprendizado de Máquina e é amplamente utilizado acadêmica e comercialmente.

O WEKA disponibiliza cinco modos diferentes de interação:

W	MONITOR	LCD	19	ACER	X193W	WIDE	ABD	20000:1
o_1	1	1	1	1	1	1	0	0
o_2	1	1	1	1	1	0	1	1

Tabela 2.1: Representação vetorial das palavras

PUC-Rio - Certificação Digital Nº 1012636/CA

Interface Gráfica: Explorer Através desta interface, é possível escolher um *dataset*, aplicá-lo um conjunto de filtros, e executar os algoritmos de classificação, agrupamento ou associação sobre as suas instâncias.

Interface Gráfica: KnowledgeFlow Permite realizar as mesmas operações descritas acima, mas fornece uma interface intuitiva, onde é possível desenhar um fluxo de execução mais complexo (*workflow*), através de elementos gráficos.

Interface Gráfica: Experimenter Simplifica a execução de vários experimentos utilizando *datasets* diferentes em cima de uma mesma configuração.

Simple CLI Console para executar comandos do WEKA.

Programaticamente O WEKA fornece uma vasta API que nos permite desenvolver aplicações utilizando-se de todos os seus recursos.

Apesar da grande variedade de meios de interagir com o WEKA, o desenvolvedor que está construindo aplicações sobre sua API possui poucas ferramentas para ajudá-lo nesta tarefa, e nenhuma das interfaces gráficas disponibilizadas no ambiente do WEKA possuem o propósito de apoiarem esta atividade. E é neste ponto que o *EasyLearn* é útil, provendo uma camada de configuração de simples entendimento e facilmente reaproveitável. A proposta do *EasyLearn* é semelhante ao conceito da interface *KnowlegdeFlow* - onde é possível definir um *workflow* que será usado para guiar a execução do WEKA, no entanto, seu foco é integração entre aplicação, e não é baseada em elementos visuais mas em arquivos XML com um padrão bem definido.

O *EasyLearn* foi construído sobre o WEKA, conforme explicamos no capítulo 4, contudo, prevemos como trabalho futuro (capítulo 6) modificar sua arquitetura para deixá-lo capaz de suportar outros *frameworks* de Aprendizado de Máquina, como o *Apache Mahout*, *Encog Machine Learning Framework*, *GraphLab*, *RapidMiner*, etc.

PUC-Rio - Certificação Digital Nº 1012636/CA

3
Processos de Aprendizado Supervisionado

Dedicamos um capítulo para dissecar os processos de Aprendizado Supervisionado, mais precisamente, aqueles que tratam o problema da Classificação (seção 2.1). Os processos descritos aqui foram utilizados como base para o desenvolvimento do *framework* proposto. No entanto, o *framework* foi implementado para suportar variações nestes fluxos - como o procedimento descrito em (Kotsiantis07), sem que haja necessidade de mudanças no código-fonte. Além disto, novos processos podem ser definidos e implementados sem necessitar de conhecimentos avançados de programação.

Para facilitar o entendimento desenhamos cada processo descrito neste capítulo utilizando a notação *BPMN (Business Process Modeling Notation)*.

3.1
Pre-requisitos

Todos os processos de Aprendizado Supervisionado descritos abaixo, têm como pre-requisito a construção de um Corpus. A etapa da construção do Corpus pode ser vista em detalhe no capítulo 5.2, onde explicamos como construímos o Corpus utilizado no estudo de caso de Classificação de Produtos do comércio eletrônico.

3.2
Avaliação

O primeiro processo é o de Avaliação (figura 3.1). Normalmente este processo é o mais dispendioso em relação ao tempo e ao esforço investido, pois, muitas vezes, são necessárias inúmeras execuções deste processo, até que seja alcançado uma acurácia satisfatória. A cada execução diversos parâmetros precisam ser calibrados até que tudo esteja pronto para o processo seguinte: aprendizado (seção 3.3).

Extrair/Filtrar Corpus Uma vez que o Corpus tenha sido criado, o primeiro passo é transformá-lo em um DataSource (já utilizando a nomenclatura do *EasyLearn*). Um DataSource é um arquivo de entrada de dados, independente do formato, que pode conter todo o Corpus ou um subconjunto

PUC-Rio - Certificação Digital Nº 1012636/CA

dele. No estudo de caso, apresentado no capítulo 5, dividimos o nosso Corpus em vários DataSources: um para cada categoria.

Processar dados Normalmente é necessário um processamento à priori do DataSource, isto é, um pré-processamento. Esta etapa de preparação dos dados possui um papel fundamental para a qualidade dos resultados obtidos ao final do processo de avaliação, pois é onde aplicamos uma série de técnicas de padronização, simplificação e correção da entada a ser processada. Na seção 3.5 exemplificamos e descrevemos alguns artifícios utilizados nesta etapa.

Realizar / Aperfeiçoar Engenharia de Atributos O passo subsequente ao pré-processamento é criar um modelo de atributos baseado no domínio do problema em questão, afinal, nem sempre o DataSource irá conter todos os atributos necessários para capturar os detalhes do problema a ser solucionado. Neste caso, é necessário identificar e extrair atributos ocultos do modelo inicial através da análise de atributos mais primitivos (Markovitch02). Para ajudar nesta etapa o *EasyLearn* provê alguns recursos para extração de atributos (capítulo 4). Assim como, eventualmente, é necessário criar novos atributos para enriquecer o modelo, algumas vezes precisaremos restringir a quantidade de atributos removendo atributos redundantes e irrelevantes para melhorar a eficiência dos classificadores (Yu2004). Para a seleção de atributos, o Weka fornece diversos Seletores que foram encapsulados no *EasyLearn*.

Separar DataSource Caso não tenhamos disponível um DataSource de Publicação, isto é, um DataSource construído especialmente para realização de testes para avaliação do modelo a ser criado, precisamos dividir o DataSource em dois conjuntos, um para o treinamento e outro para a avaliação. As técnicas de avaliação são descritas mais detalhadamente logo abaixo, no processo *Avaliar Modelos*.

Escolher e Treinar Classificadores Infelizmente, não existe um classificador que atenda a todos os propósitos. Cada classificador, normalmente, possui as suas vantagens e desvantagens. Alguns classificadores não suportam atributos textuais, ou atributos com valores nulos, outros classificadores podem não apresentar um desempenho satisfatório para um determinado DataSource. Enfim, existe uma grande variedade de Classificadores disponíveis, e um passo importante é escolher bem o conjunto dos classificadores elegíveis para solucionar o problema em questão. O *EasyLearn* tem um papel importante neste momento, pois torna possível

PUC-Rio - Certificação Digital Nº 1012636/CA

avaliar uma série de classificadores em uma única execução do fluxo. Desta forma, não é necessário executar todos os passos anteriores a este para cada classificador, isto possibilita uma economia de tempo do desenvolvedor, melhorando a eficiência da execução do processo de Avaliação. Uma vez que o conjunto de classificadores é definido, cada classificador precisa ser treinado com a parte do DataSource separada anteriormente para o aprendizado.

Avaliar Modelos Este processo é executado para cada classificador a ser avaliado e recebe como entrada dois artefatos: o classificador já treinado e o DataSource de avaliação. Existem basicamente três técnicas para medir a acurácia do modelo produzido ao final do processo de classificação. A primeira consiste em dividir o DataSource inicial em duas partes: uma parte usada no treinamento do classificador - contendo dois terços das instâncias, e a outra parte utilizada para avaliar o classificador treinado - contendo o restante das instâncias. A segunda técnica, conhecida como Validação Cruzada, consiste em dividir o DataSource em algumas partes mutualmente exclusivas e de igual tamanho, e para cada parte, o classificador é treinado utilizando a união de todas as demais partes, e avaliado utilizando as instâncias que não pertencem ao conjunto de instâncias usadas no treinamento (explicaremos melhor esta técnica aplicada ao Estudo de Caso, na seção 5.2.3). A precisão é calculada baseada na média de erro da classificação para cada subparte. Este método é computacionalmente mais custoso, no entanto, é normalmente utilizado quando precisamos ter uma medida mais precisa da acurácia (Kotsiantis07). Por fim, a terceira técnica não exige que o DataSource inicial seja dividido automaticamente, pois conta com um outro DataSource (DataSource de publicação ou de teste) construído manualmente para testar o modelo gerado. Normalmente a construção deste conjunto de Publicação requer muito esforço, e nem sempre a aplicação desta técnica é viável para o problema, por isto, as outras duas técnicas são mais utilizadas no cotidiano (Alpaydin04).

Identificar Inconsistências do corpus Até que o Corpus esteja suficientemente consistente para ser utilizado com segurança, é natural que ele passe por uma série de revisões e atualizações. Normalmente, este é um processo manual, e é uma etapa importante para melhorar a qualidade da solução.

Atualizar Corpus Cada inconsistência percebida na etapa anterior precisa ser corrigida no Corpus. Este passo normalmente acontece por fora do

PUC-Rio - Certificação Digital Nº 1012636/CA

EasyLearn, no entanto, caso haja necessidade, é possível modificar o *framework* para suportar tarefas como esta. Após atualizar o Corpus, o processo de avaliação deve ser reiniciado.

Salvar Relatórios de Execução Ao final de cada execução do processo de avaliação, o *framework* deve gerar um relatório, informando, para cada classificador, sua acurácia, o tempo gasto para treinar e o tempo gasto para avaliar todas as instâncias. Além disto, é importante que este relatório contenha informações que indiquem quais foram os parâmetros utilizados naquela execução. Utilizando o *EasyLearn* é possível configurar outros relatórios, além de definir se tais estatísticas devem ou não ser armazenadas para futuras referências.

3.3 Aprendizado

Uma vez que o processo de Avaliação tenha sido concluído, entra em cena o processo de Aprendizado (figura 3.2), que é quando preparamos e treinamos os classificadores escolhidos anteriormente para serem utilizados com dados reais.

Abrir DataSource O primeiro passo é carregar o DataSource que será utilizado para treinar os classificadores. Neste caso, podemos utilizar todas as instâncias do DataSource, já que não haverá etapa de avaliação. Na maioria das vezes, isto ajuda a melhorar a qualidade da solução. No entanto, a ideia de que quanto mais instâncias, melhor a qualidade, nem sempre é uma verdade no processo de Classificação, inclusive, em alguns casos é necessário reduzir a quantidade de instâncias, seja por questões de desempenho, ou até mesmo, por prejudicar a acurácia gerando ruído no Corpus. Então, fica a critério do usuário da *API* escolher se deverá ou não utilizar todo o DataSource no treinamento ou apenas a parte utilizada no processo de Avaliação.

Preprocessar dados Esta etapa de pré-processamento é a mesma etapa do processo de Avaliação. Para evitar que o usuário tenha que duplicar a definição utilizada para configurar o processo de Avaliação, o *EasyLearn* possui uma forma de simplificar os arquivos de configuração, evitando duplicação do código. Este artifício é explicado melhor no capítulo 4.

Realizar Engenharia de Atributos Assim como na etapa de pré-processamento, esta etapa também já foi definida no processo de Avaliação e deve ser reaproveitada neste momento.

PUC-Rio - Certificação Digital Nº 1012636/CA

Treinar Classificadores Cada classificador utilizado deve ser treinado com o DataSource que foi carregado no primeiro passo deste processo.

Serializar e persistir Modelos Ao final do treinamento, cada classificador é serializado e persistido para ser utilizado no processo seguinte (seção 3.4). A forma que o *EasyLearn* persiste os classificadores pode ser redefinida. Atualmente, o *framework* apenas serializa e comprime cada classificador utilizando *Deflate* ou *ZIP*, mas poderíamos salvar os classificadores criptografados, caso haja a necessidade.

3.4 Classificação

O processo de Classificação (figura 3.3) é o ultimo processo do ciclo de Aprendizado Supervisionado por Classificação. Caso, ao final deste ciclo, os resultados não sejam satisfatórios, é necessário voltar à primeira etapa e reajustar cada parâmetro da execução. Este processo ilustra como, enfim, a solução gerada é utilizada em um contexto real.

Extrair Dados Reais Esta é a etapa de carga do DataSource contendo os dados reais a serem classificados.

Preprocessar dados O DataSource contendo os dados reais, então, deve ser pré-processados da mesma forma que os DataSources de treinamento e avaliação foram processados nos processos anteriores, caso este cuidado não seja tomado podemos resultar em um DataSource incompatível com o modelo persistido no processo de Aprendizado.

Carregar Modelo Serializado O modelo treinado e serializado no processo de Aprendizado deve ser carregado em memória.

Classificar DataSource O DataSource contendo os dados reais é classificado pelo modelo carregado no passo anterior.

Processar DataSource Classificado O resultado da classificação é então pós-processado. Esta etapa é um *HotSpot* do *framework* concebido, e deve ser implementada pelo usuário da *API*. Normalmente esta fase consiste em salvar as instâncias classificadas em um sistema legado, ou em um arquivo tabulado. Também pode ser interessante classificar cada instância de acordo com a certeza do classificador, permitindo assim, que instâncias que possuem um grau baixo de certeza em sua classificação possam ser revisadas manualmente ao final do processo de Classificação.

PUC-Rio - Certificação Digital Nº 1012636/CA

3.5 Procedimentos de Pré-processamento

Mencionamos abaixo as técnicas mais corriqueiras de Processamento de Linguagem Natural, Mineração de Texto, Recuperação de Informação (*Information Retrieval*) e Aprendizado de Máquina que suportam a etapa de pré-processamento do DataSource. Parte desta seção foi baseada no livro de (Indurkhya10) que dedica um capítulo inteiro para explicar os desafios do pre-processamento textual.

Encoding Em domínios de problemas que envolvem textos em diversas línguas, por exemplo, configurar e padronizar o *Encoding* do DataSource pode ser um desafio, mas é um dos primeiros, senão o primeiro passo, antes de aplicar qualquer outro filtro de pré-processamento.

Normalização Este é um conceito amplo, e, neste caso, estamos nos referindo à todas as formas de normalização possíveis. **Normalização textual:** processamento das *"non-standard words"* (NSW), tais como endereços de e-mail, URL, abreviações, valores monetários, datas (Sproat01), espaços em branco, caracteres especiais e palavras com diferenças na capitalização (WinNT, LaTeX) (Manning99), hifenização, etc...; **Normalização Unicode:** procedimento responsável por substituir sequências de caracteres compostos (letras acentuadas, cedilha, etc..) - que podem possuir diversas representações possíveis, dependendo da codificação, em uma única sequência, evitando que o mesmo carácter composto seja representado de várias formas (Unicode11).

Stemming *Stemming* é uma forma de normalização e consiste no processo de reduzir uma palavra ao seu radical, removendo os seus afixos (Manning99). Por exemplo:

$$Stemming('pedrada') = Stemming('pedrinha') = 'pedr'.$$

Através do *Stemming* é possível reduzir a quantidade de atributos do modelo gerado, uma vez que várias palavras que compartilham o mesmo radical serão substituídas pelo seu radical comum. Esta técnica é útil para melhorar a eficácia dos algoritmos de busca textual e é amplamente utilizada na área de Processamento de Linguagem Natural.

Verificação Ortográfica (*Spellchecking*) Estimativas indicam que a frequência dos erros de digitação variam entre 0.05% em matérias jornalísticas revisadas, até 38% para aplicações complexas, como busca em um catálogo telefônico, por exemplo (Kukich92).

PUC-Rio - Certificação Digital Nº 1012636/CA

Para mitigar este erro e evitar que ele cause distorções nos resultados, é recomendado, na maioria das tarefas de Processamento de Linguagem Natural, aplicar uma verificação ortográfica antes de iniciar o processamento.

No trabalho de (Kukich92) e nos livros de (Indurkhya10) e (Teller00) podemos conferir em detalhes as técnicas mais utilizadas para correção ortográfica, tais como Distância Mínima de Edição, técnicas de análise de *N-grams*, técnicas probabilísticas ou baseadas em regras, etc..

Substituição de Sinônimos Dependendo do problema a ser resolvido, pode ser relevante substituir palavras ou sentenças com o mesmo significado por seus sinônimos a fim de reduzir o vocabulário a ser tratado. Para sinônimos da língua inglesa, esta tarefa pode ser apoiada com o uso do WordNet - que é um grande banco de dados léxico do Inglês. No WordNet, substantivos, verbos, adjetivos e advérbios são agrupados em conjuntos de sinônimos cognitivos (*synsets*), cada um expressando um conceito distinto. Os *synsets* estão interligados por meio de relações semânticas (Fellbaum98).

Segmentação O processo de segmentação pode se referir a palavras ou sentenças em um documento textual. A segmentação de palavras visa quebrar uma sequência de caracteres de acordo com o limite de cada palavra. Cada palavra identificada é chamada de *token*, e este procedimento também é conhecido como *Tokenização*. Na maioria das linguagens este processo funciona baseado em espaços em branco que comumente são utilizados para separar as palavras, contudo, em algumas línguas/dialetos - especialmente as dos países do leste asiático (chinês, japonês, tailandês), o processo de segmentação de palavras pode ser bem desafiador, já que, neste caso, não é possível realizar a separação das palavras por espaços em branco exigindo abordagens mais sofisticadas (Manning99).

Eliminação de Stopwords As *stopwords* são termos frequentes em uma linguagem e, geralmente, não são representativos no contexto do documento analisado. Esta lista é constituída por artigos, advérbios, números, pontuação, preposições e pronomes.
Exemplos de *stopwords* da língua portuguesa : *a, alguma, após, com, de, e, é, ela, ele, em, o, os, ou, pela, porque, tem, um, uma*, entre outras.
A remoção das *stopwords* pode oferecer um ganho de desempenho para os algoritmos de classificação, dependendo do caso.

PUC-Rio - Certificação Digital Nº 1012636/CA

Seleção de Atributos A seleção de atributos é apresentada na seção 2.5 e foi mencionada nesta lista para ressaltar que se trata de uma técnica executada à priori.

Seleção de Instâncias A habilidade de analisar e interpretar grandes conjuntos de dados está muito aquém da capacidade de armazenamento atual. Para confrontar este desafio, as áreas de Descoberta de Conhecimento e Mineração de Dados precisam estar em constante evolução para permitir lidar com *datasets* cada vez maiores. O tópico de Seleção de Instâncias compreende uma série de técnicas, tais como detecção de inconsistências no corpus (como visto no próximo item *Detecção de ruídos*) e redução da quantidade de instâncias do corpus - através de métodos de otimização, onde buscamos minimizar a quantidade de instâncias e maximizar a precisão (Motoda01). Um exemplo real de seleção de instâncias é o caso de reconhecimento de escrita manual - ilustrado logo na apresentação dos conceitos básicos de Classificação (seção 2.1), onde precisamos classificar um conjunto de coordenadas cartesianas em um algarismo de zero à nove. Dependendo da forma que estas coordenadas tenham sido capturadas (por sensores ou através de dispositivos sensíveis ao toque, por exemplo) é normal que haja uma distância muito pequena entre duas coordenadas (de acordo com a frequência de leitura dos sensores), resultando na leitura de um número de coordenadas muitas vezes maior do que a quantidade necessária para a classificação. Caso esta quantidade exagerada de coordenadas esteja impactando no desempenho da classificação recomenda-se realizar uma suavização dos dados, isto é, a seleção de pontos chaves da imagem capturada (conjunto de coordenadas). Desta forma, a tendência é reduzir a quantidade de ruídos e fenômenos pontuais ou momentâneos do DataSource.

Detecção de ruído (*outlier*) Esta técnica possui diversas denominações na literatura: detecção de anomalias, detecção de inconsistências, detecção de desvio, mineração de exceções, etc. (Hodge04). De acordo com (Barnett94), um *outlier* pode ser definido como:

> Uma observação (ou subconjunto de observações) que parece ser inconsistente em relação ao restante do conjunto de dados.

Como estamos na seção de pré-processamento, nos referimos neste item à detecção de ruídos como ferramenta de processamento à priori, uma vez que o termo de detecção de *outliers* também é utilizado para se referir a uma gama de problemas de classificação, por exemplo, à sis-

PUC-Rio - Certificação Digital Nº 1012636/CA

temas de detecção de intrusão (através de análise de arquivos de *log* a procura de atividades suspeitas), ou sistemas de detecção de fraudes (através da análise de uma série de operações de cartão de crédito). No nosso contexto, estamos analisando a detecção de *outliers* como artifício para melhorar a qualidade do conjunto de dados - eliminando as inconsistências e minimizando possíveis interferências nos resultados do processo de classificação. Existem basicamente três formas de realizar este procedimento. Duas delas utilizam métodos de Aprendizado Supervisionado, isto é, pode demandar um grande esforço de preparação de um segundo corpus indicando e classificando manualmente o que é um *outlier* - tarefa que, dependendo do caso, provavelmente não compense o tempo investido em relação ao benefício gerado, no entanto, também é possível cumprir este propósito utilizando aprendizado não-supervisionado, mais especificamente, com técnicas de Agrupamento (*clustering*). Este último artifício possui um custo muito menor em relação aos métodos de aprendizado supervisionado. Para se aprofundar mais neste assunto, recomendamos a leitura do trabalho *A Survey of Outlier Detection Methodologies* (Hodge04).

Instâncias Sintéticas Ao passo que eventualmente precisamos reduzir o tamanho do nosso conjunto de dados - como visto no item *Seleção de Instâncias*, em alguns casos, é necessário que instâncias produzidas artificialmente sejam introduzidas no corpus. Isto pode ser necessário quando estamos trabalhando com *datasets* desbalanceados, ou seja, quando existe uma discrepância alta em relação à distribuição de frequência das classes presentes no *dataset* analisado. *Datasets* com esta características podem interferir negativamente no processo de classificação, podendo produzir palpites tendenciosos para as classes que possuem uma representatividade maior. Com o objetivo de atingir uma distribuição mais igualitária de frequências das instâncias para classes, em (Kegelmeyer02), é apresentado uma técnica que visa produzir instâncias sintéticas para as classes com menor representatividade, enquanto, elimina algumas instâncias das classes que possuem maior distribuição.

Um exemplo de contexto onde o uso desta técnica é recomendado é na análise de mamografias para a detecção automática de tumores. Em um *dataset* de mamografias, em torno de 98% dos exames são normais, enquanto apenas 2% das mamografias contém anomalias. Uma estratégia convencional de classificação, deve produzir uma acurácia próxima aos 98%, mas não garante uma taxa de acerto elevada para a classe minoritária. A natureza desta aplicação requer uma taxa de erro elevada

PUC-Rio - Certificação Digital Nº 1012636/CA

para os exames contendo anomalias, enquanto permite uma taxa de erro pequena na predição da classe majoritária - onde não há anomalias (Woods93).

Atributos ausentes (*Missing Features*) Muitos algoritmos de classificação não suportam atributos com valores ausentes. Ao utilizar estes classificadores, precisamos ter o cuidado de tratar os atributos sem valores no nosso *dataset* substituindo-os, sempre que possível, por valores plausíveis - este procedimento é chamado de imputação. Caso este tratamento não seja feito cuidadosamente, podemos afetar nossos classificadores deixando-os mais inclinados à realização de predições erradas. Para se aprofundar neste assunto, encontramos uma análise de quatro procedimentos para tratar atributos ausentes em (Batista03) .

PUC-Rio - Certificação Digital Nº 1012636/CA

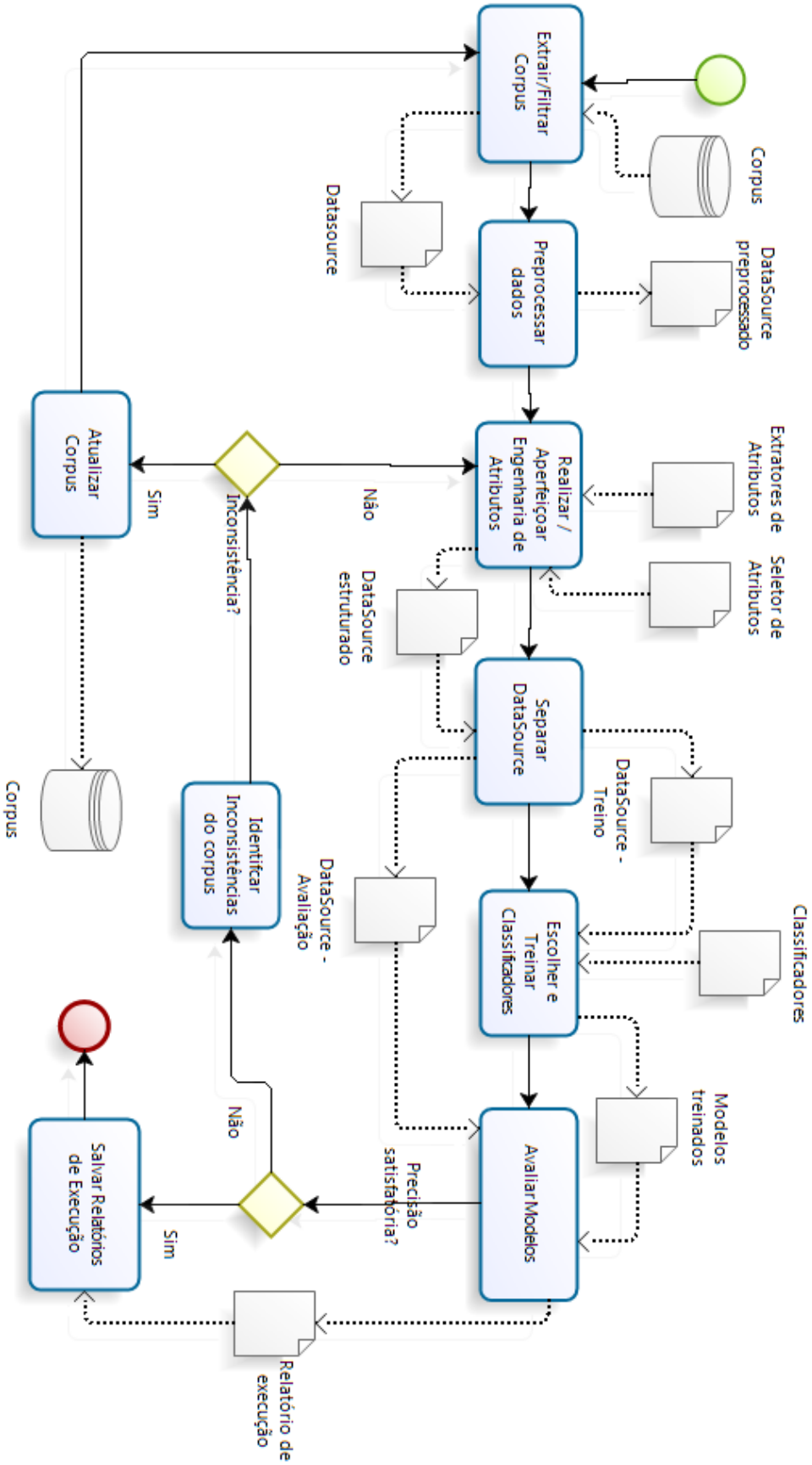

Figura 3.1: Processo de avaliação

PUC-Rio - Certificação Digital Nº 1012636/CA

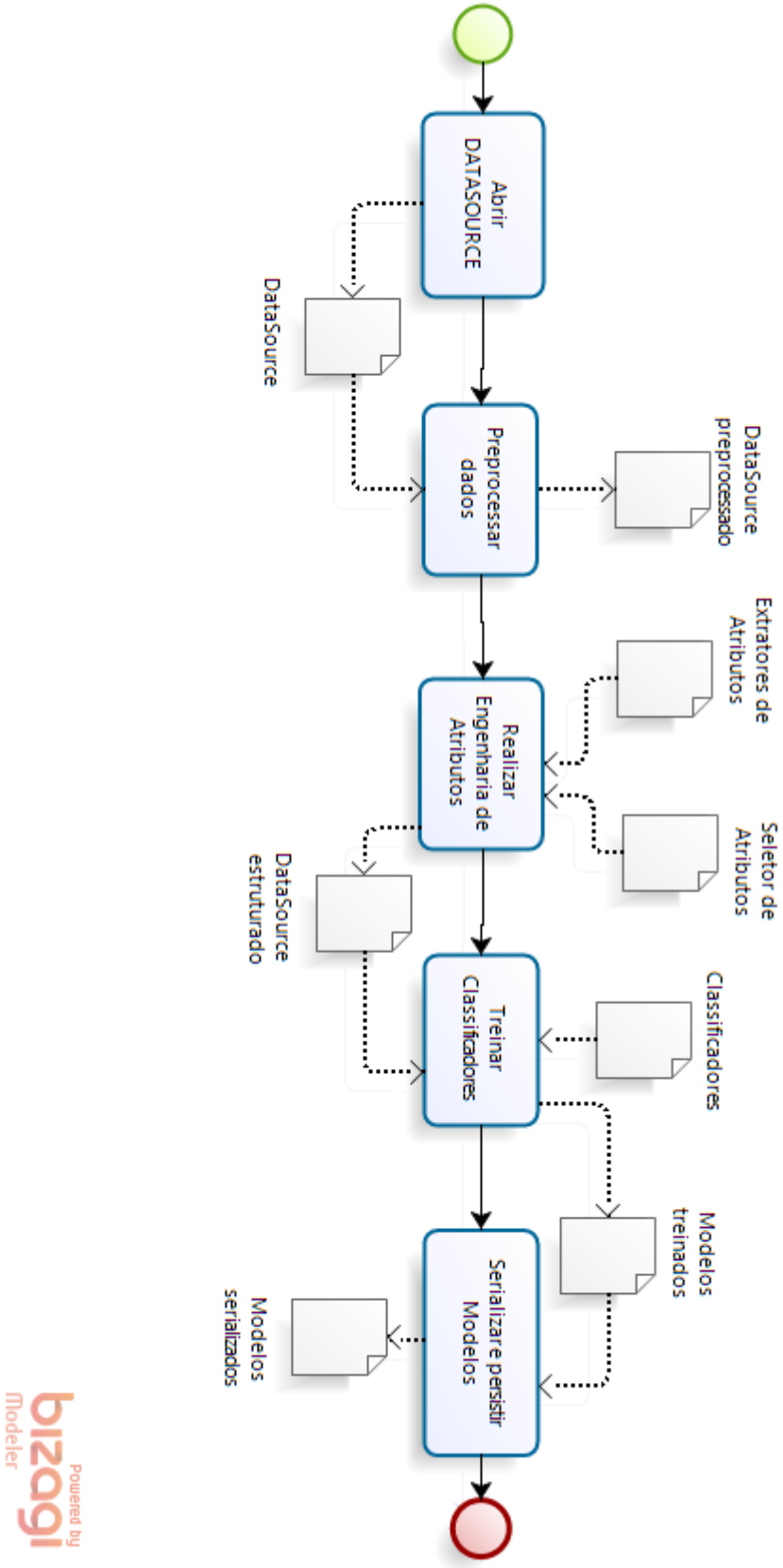

Figura 3.2: Processo de aprendizado

PUC-Rio - Certificação Digital Nº 1012636/CA

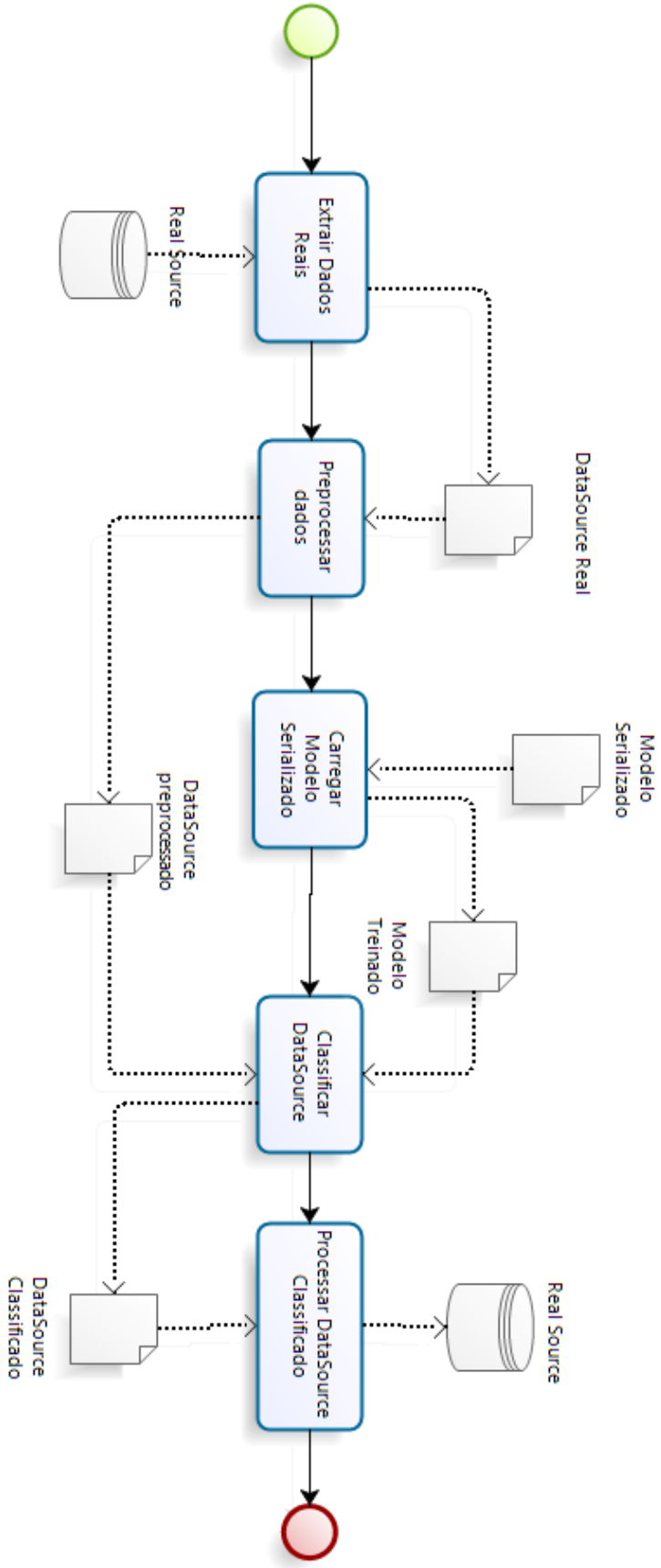

Figura 3.3: Processo de classificação

PUC-Rio - Certificação Digital Nº 1012636/CA

4
Arquitetura

A arquitetura apresentada neste capítulo foi definida para cumprir os requisitos enumerados abaixo (seção 4.1). O *EasyLearn* se apoia sobre o *framework WEKA* da Universidade de Waikato, criando uma camada intermediaria para o desenvolvedor de aplicações para classificação de entidades, como é ilustrado pela figura 4.1.

Todos os classificadores e filtros implementados pelo *WEKA* podem ser facilmente encapsulados para serem utilizados pelo *EasyLearn*.

O *EasyLearn* recebe como entrada um conjunto de arquivos de configuração no formato XML, além do DataSource a ser processado. Sua saída é customizável, e pode ser configurada para produzir, por exemplo, relatórios de acurácia da classificação, o próprio DataSource classificado, ou o modelo de classificação já treinado.

4.1
Requisitos

Relacionamos os seguintes requisitos para pautar o desenvolvimento do *EasyLearn*:

- Simples entendimento, manutenção e configuração.
- Comportar os processos de Avaliação, Aprendizado e Classificação descritos no capítulo 3 e suas possíveis variações.
- Suportar um fluxo genérico de execução, atendendo a qualquer problema relacionado ao Aprendizado Supervisionado utilizando Classificação.
- Suportar e combinar vários seletores e classificadores em uma mesma execução.
- Suportar qualquer tipo de entrada de dados (JDBC, ARFF, CSV...).
- Ser facilmente integrável com sistemas legados.
- Produzir relatórios de execução por rodada.
- Armazenar organizadamente os modelos produzidos, salvando o estado de cada classificador após o treinamento, para tornar os processo de avaliação e classificação mais eficientes.

PUC-Rio - Certificação Digital Nº 1012636/CA

– Prover um repertório dos principais artifícios de mineração de texto, processamento de linguagem natural e outros recursos comuns à maioria dos problemas relacionados.

– Utilizar o *framework* WEKA como motor de Aprendizado de Máquina de forma transparente para o usuário da *API*.

4.2 Conceitos

Definimos os conceitos de Base de Conhecimento (*KnowledgeBase*) e de Fluxo de Conhecimento (*KnowlegeFlow*) para apoiar a implementação do *framework*, ambos os conceitos são explicados nas seções seguintes.

4.2.1 Knowledge Base

A ideia por trás da *KnowledgeBase* é fornecer um espaço de trabalho (*workspace*) para o *EasyLearn*. A Base de Conhecimento é onde ficam armazenados os relatórios de execução do framework (knowledgeBase/reports), dicionários utilizados pelos extratores de atributos (knowledgeBase/libary), os modelos serializados (knowledgeBase/classifier), os DataSources utilizados - quando os dados não vêm diretamente do Banco de Dados (knowledgeBase/source) e, por fim, os fluxos de conhecimento descritos na seção 4.2.2 (raiz da knowledgeBase).

4.2.2 Knowledge Flow

O fluxo de conhecimento é definido por um arquivo de configuração que determina a sequência de execução do *EasyLearn*. Este conceito sustenta o requisito de flexibilidade do framework. Através da criação de um fluxo de conhecimento é possível estabelecer de uma forma mais amigável os processos definidos no capítulo 3. Além disto, o *EasyLearn* permite criar uma hierarquia entre os fluxos de conhecimento de uma forma semelhante às linguagens orientadas à objeto, implementando os conceitos de herança e polimorfismo, maximizando o reuso dos arquivos de configuração para problemas mais complexos - estas propriedades são explicadas em detalhes na seção 4.4.

PUC-Rio - Certificação Digital Nº 1012636/CA

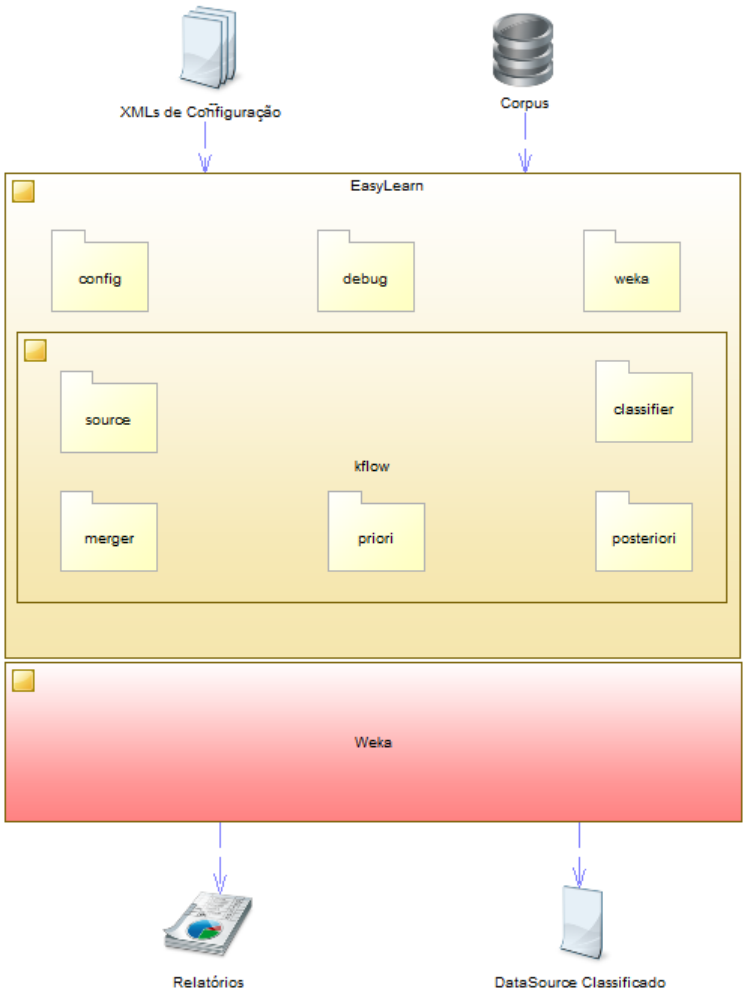

Figura 4.1: Visão geral da Arquitetura do framework

4.3 Pacotes e Classes do EasyLearn

No anexo A descrevemos cada pacote visto na figura 4.1 juntamente com as suas classes, detalhadamente. Estes elementos constituem a base do *framework EasyLearn*

4.4 Configuração

Uma das principais premissas por trás do *EasyLearn* é simplificar o desenvolvimento de projetos de classificação de entidades, no entanto, cada solução proposta para cada problema pode possuir inúmeras possibilidades de customização e parametrização. Para manter a simplicidade do *framework* e, ainda assim, atingir todos os requisitos de flexibilidade da solução, criamos duas formas de configuração da infra-estrutura proposta:

– Programaticamente (objetos do pacote `Matcher.Config`)

PUC-Rio - Certificação Digital Nº 1012636/CA

– XML de configuração

Explicamos os objetos do pacote `Matcher.Config` na seção A.2. A seção atual foi dedicada para ver com mais detalhes a estrutura do XML de configuração - que é o coração do *framework*.

Na figura 4.2 podemos ver o diagrama do esquema de definição do XML de configuração utilizado pelo *EasyLearn* (XSD). Após o processamento do arquivo de configuração, cada elemento do XML é transformado em objetos do pacote `Matcher.Config`. Sendo assim, é possível identificar de forma clara a relação de cada bloco interno do XML com as classes explicadas na seção A.2.

A principal ideia pro trás do conceito de `KnowledgeFlow` (seção 4.2.2) é permitir o mapeamento de cada um dos processos descritos no capítulo 3. Desta forma, cada XML contém a configuração de um `KnowledgeFlow`, que, por sua vez, representa um processo de aprendizado supervisionado. Contudo, se analisarmos com atenção os processos estudados no capítulo 3, podemos perceber que estes processos possuem algumas interseções em seus fluxos (como por exemplo a atividade **Pré-processar dados**). Caso não nos atentássemos a estas operações comuns entre os processos de aprendizado supervisionado, introduziríamos um problema para manter os arquivos de configuração, uma vez que, qualquer modificação em um `KnowledgeFlow` poderia afetar e repercutir em vários outros arquivos - denegrindo a capacidade de manutenção das aplicações que utilizassem o *framework*.

Para solucionar este problema, buscamos inspiração nas linguagens orientadas a objeto (OO) que possuem como um dos principais objetivos facilitar o reaproveitamento de código e de componentes de software (Larman04). Projetamos o `KnowledgeFlow` tomando emprestado do universo OO os conceitos de `herança` e `polimorfismo`. Através da herança é possível criarmos estruturas de configuração com hierarquias complexas e, ainda assim, manter cada XML de configuração inteligível. Com o uso do polimorfismo, podemos redefinir comportamentos determinados no XML raiz. Esta combinação de conceitos se mostrou muito robusta para a resolução de problemas complexos, e nos possibilitou atingir os requisitos de simplicidade e flexibilidade da solução.

PUC-Rio - Certificação Digital Nº 1012636/CA

4.5 Diagrama XSD

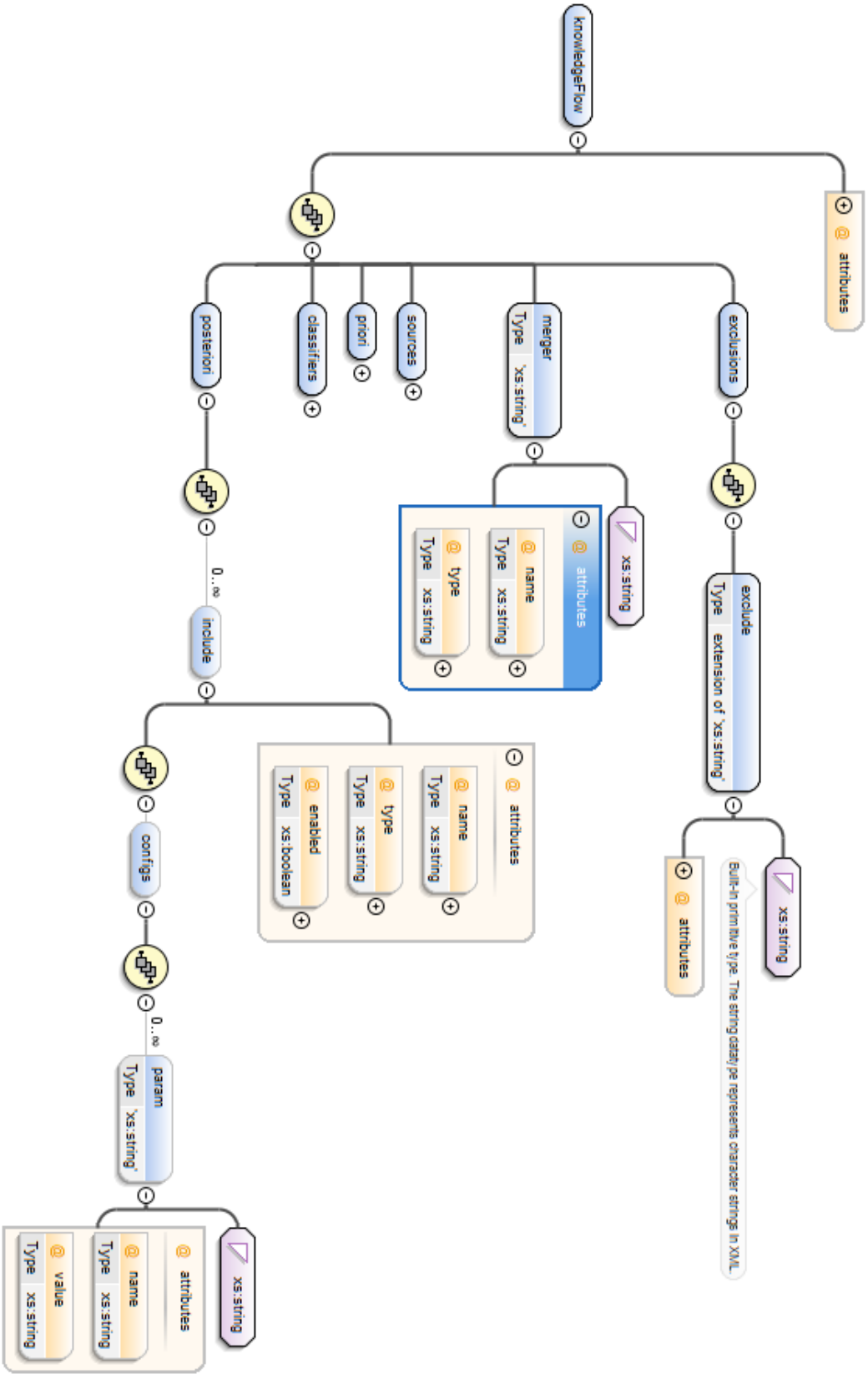

Figura 4.2: XSD - KnowledgeFlow

PUC-Rio - Certificação Digital Nº 1012636/CA

4.6 Elementos do XML de configuração

Abaixo dissecamos cada elemento do XSD (figura4.2):

O nó `KnowledgeFlow` representa todo o fluxo de execução, e possui os seguintes atributos:

name Nome do `KnowledgeFlow` a ser configurado.

extends Permite especificar o nome do XML raiz (implementação da herança).

flow Permite definir a classe que irá controlar o fluxo de execução, caso seja necessário um fluxo de execução alternativo (deve herdar da classe `MatcherFlowController`).

O nó `KnowledgeFlow` possui sete blocos internos:

configs Permite passar parâmetros do tipo `<param>` para o `FlowController`.

sources Define o conjunto de *DataSources* que será utilizado ao longo da execução do fluxo, cada *DataSource* é adicionado pelo comando `<include>` (pacote `Matcher.KFlow.Source`, seção A.4).

priori Define os pré-processadores que serão utilizados para filtrar os *DataSources*, cada pré-processador é adicionado pelo comando `<include>` (pacote `Matcher.KFlow.Priori`, seção A.5).

merger Permite definir o `InstanceMerger` através do atributo `type` que deverá apontar para a classe a ser utilizada para a manipulação do `ClassificationContext` (pacote `Matcher.KFlow.Merger`, seção A.7).

classifier Especifica os componentes que serão utilizados para classificar as instâncias. Cada classificador é adicionado pelo comando `<include>` (pacote `Matcher.KFlow.Classifier`, seção A.6).

posteriori Define os elementos que serão utilizados para processar a saída do *framework*. Cada pós-processador é adicionado pelo comando `<include>` (pacote `Matcher.KFlow.Posteriori`, seção A.8).

exclusions Permite suprimir comportamentos de elementos definidos em níveis mais elevados na hierarquia dos arquivos de configuração. Cada exclusão pode ser definida pelo comando `exclude`.

PUC-Rio - Certificação Digital Nº 1012636/CA

Os blocos de configuração `<sources>`, `<priori>`, `<classifier>` e `<posteriori>`, utilizam-se do comando `<include>` para adicionarem seus elementos. Relacionamos abaixo os atributos da *tag* `<include>`:

name Nome do elemento a ser incluído.

type Classe a ser instanciada para o elemento incluído (deve herdar de `MatcherPart`)

enabled Permite habilitar ou desabilitar o elemento a ser incluído, facilitando os testes.

before Permite inserir o elemento corrente antes do elemento especificado por este atributo. Este atributo e utilizado para definir ou modificar a ordem de execução de cada elemento, em relação à toda a hierarquia de configuração).

configs:param Lista de parâmetros de configuração do tipo `<param>`.

PUC-Rio - Certificação Digital Nº 1012636/CA

4.7 Exemplos de configurações

Para exemplificar o uso do XML de configuração do *EasyLearn*, elencamos o problema de classificação de e-mails em `SPAM` e `NÃO SPAM` pela sua simplicidade. Cada solução de Classificação precisa implementar os três processos mencionados no capítulo 3.

Para o exemplo abordado, o DataSource de entrada (*RawSource*) será um arquivo ARFF (Weka08) contendo duas colunas:

@relation SpamCorpus

@attribute class {spam, email}
@attribute content **string**

Figura 4.3: Estrutura do arquivo de entrada ARFF.

4.7.1 SpamParent

Como alguns processos possuem interseções em relação às suas operações, recomendamos criar um arquivo de configuração `raíz` contendo as partes comuns aos processos, com este propósito criamos o arquivo `SpamParent` (listagem 1). Este arquivo contém a lista de pre-processadores (`<priori>`) que serão utilizados por todos os arquivos XML que herdam de `SpamParent`, além dos classificadores (`<classifiers>`) e dos elementos de pós-processamento (`<posteriori>`).

Elementos do XML raíz:

- **priori**
  1. StringNormalizer: padronização da entrada.
  2. ExtractorPlaceHolder: facilita a inclusão de novas partes pelos XML filhos. Facilitando a extração de atributos, por exemplo, antes do processo de *tokenização*.
  3. StringTokenizer: Cria a matriz de atributos (bag of words) para o campo `content` do DataSource.
  4. DatasetSplitter: Separa o DataSource em dois.
- **classifiers**
  1. J48: algoritmo de classificação utilizando árvore de decisão.
  2. SMO: algoritmo de classificação utilizando SVM.
- **posteriori**
  1. CrossValidateEvaluateClassifier: executa a validação cruzada 2-folds sobre o DataSource completo.
  2. EvaluateClassifier: realiza outra medida de avaliação de precisão com o DataSource `EvalSource`.

PUC-Rio - Certificação Digital Nº 1012636/CA

4.7.2 SpamEval

O arquivo `SpamEval` (listagem 2) é responsável por implementar o processo de avaliação descrito na seção 3.2 e herda do arquivo `SpamParent`.

Elementos do XML SpamEval:

– **source**

1. RawSource: arquivo de entrada ARFF definido na listagem 4.7.

– **priori**

1. Synonym: substitui os sinônimos do corpo do email (content) por palavras com o mesmo significado. Incluído antes do ExtractorsPlaceHolder definido no arquivo `SpamParent`.
2. MixedWordModel: substitui palavras misturadas com números por suas correspondentes, contendo apenas letras (ex: `V1AGRA -> VIAGRA`).

4.7.3 SpamLearn

O arquivo `SpamLearn` (listagem 3) é responsável por implementar o processo de aprendizado descrito na seção 3.3 e herda do arquivo `SpamEval`, omitindo alguns comportamentos (`<exclusions>`).

Elementos do XML SpamLearn:

– **exclusions** Todos os elementos relativos ao processo de avaliação são excluído do processo de aprendizado.

– **posteriori**

1. RenameSource: renomeia o DataSource completo (RawSource) para *LearnSource*, de modo a poder ser utilizado pelos classificadores declarados no arquivo `SpamParent`.
2. SerializeClassifier: persiste os classificadores na base de conhecimento (`KnowledgeBase`), com o nome de `SpamModel.<nome do classificador>` (o nome do classificador é utilizado como extensão do arquivo).

PUC-Rio - Certificação Digital Nº 1012636/CA

4.7.4 SpamClassify

O arquivo `SpamClassify` (listagem 4) é responsável por implementar o processo de classificação descrito na seção 3.4.

Elementos do XML SpamClassify:

– **merger** FilterMerger padrão, utilizado para recuperar o `MultiFilter` persistido na KnowledgeBase.

– **sources**

1. RawSource: carrega o conjunto de emails reais para serem classificados.

1. SMO: utiliza o SMO já treinado para classificar os dados do RawSource.

PUC-Rio - Certificação Digital Nº 1012636/CA

5
Estudo de Caso: Classificação Ofertas de Produtos

5.1
Requisitos

Para cumprir com sucesso o caso de classificação de produtos, o framework desenvolvido deve ser capaz de endereçar as seguintes questões relativas ao domínio do problema escolhido:

- Aprender a classificar produtos do ecommerce através de descrições de ofertas.
- Manter um modelo de atributos por categoria.
- Permitir que o modelo aprendido seja atualizado periodicamente, possibilitando correções e inserções de novas classes.
- Selecionar os atributos mais relevantes para cada categoria.
- Facilitar a escolha dos melhores classificadores para cada ocasião.
- Prover ferramental para corrigir o texto das descrições, caso haja algum erro de digitação.
- Acompanhar e guardar os resultados das medições de forma organizada, por categoria.
- Permitir gerar e gerenciar novas configurações para os *datasets* de treinamento e avaliação. Desta forma, poderíamos ter **N** *datasets* por categoria, melhorando a precisão das medições.

5.2
Corpus

Como visto em (Kotsiantis07), o primeiro passo para resolver qualquer problema de Aprendizado Supervisionado é a obtenção do Corpus. Isto é, a coleta das informações disponíveis acerca do problema a ser tratado. Além disto, a acurácia da solução gerada está fortemente ligada à qualidade do conjunto de dados. Neste capítulo discutimos os procedimentos e cuidados empregados na construção do corpus que foi utilizado ao longo dos experimentos apresentados nos capítulos seguintes.

PUC-Rio - Certificação Digital Nº 1012636/CA

5.2.1 Construção

Criação da base de dados para aprendizado de máquina é geralmente uma tarefa árdua e demorada, pois, quase sempre, é necessária intervenção manual para a construção dos gabaritos.

Contudo, neste caso, alguns comparadores de preço resolvem bem este problema para fins comerciais (não sabemos se eles resolvem de forma automática). Para automatizar o processo de construção do Corpus, criamos um *web scraper*, isto é, um robô para percorrer algumas páginas da internet em busca de informações para compor o *dataset*. Apesar de simplificar o processo, nos deparamos com alguns desafios durante a execução do *scraper*, já que a maioria dos web sites fornecem muitos entraves e barreiras, tornando este processo demorado. Relacionamos abaixo alguns dos problemas enfrentados:

- Grande demanda de banda, memória e CPU
- Bloqueio de IP no servidor alvo pela alta quantidade de requisições
- CAPTCHA anti-robôs
- Ausência de padrões no HTML entre as categorias
- Hierarquia complexa, com categorias, produtos e ofertas separadas por páginas.
- Javascript e AJAX que dificultam o processamento do HTML
- Inconsistências nos dados, como categorias duplicadas e produtos mal classificados.

Desta forma, nosso corpus foi construído em cima das listas de ofertas presentes em alguns web sites brasileiros de comparação de preço. Estas listas são divididas por categorias de produtos, como apresentado na figura 5.1.

Para cada categoria existe uma lista de produtos associados (SKU). Esta lista de produtos é uma camada abstrata, criada pelos sites de comparação de preço para agregar ofertas de um mesmo produto. Na figura 5.2, mostramos um produto da categoria Televisão. A descrição
"`Sony Bravia KDL-40BX425 LCD Plana 40 Polegadas`"é como o site de comparação de preços apresenta tal produto. Note que esta descrição diz respeito a uma classe de produto que pode ter descrições diferentes nos sites das lojas virtuais.

Chamaremos o produto da camada de SKU, dos sites de comparação de preços, de *"classe de produto"*, e sua descrição de *"descrição da classe"*. Para cada classe de produto existe uma lista de ofertas referentes a este produto nas lojas virtuais, como na figura 5.3. Cada produto comercializado por uma loja

PUC-Rio - Certificação Digital Nº 1012636/CA

Figura 5.1: Categorias dos Produtos

virtual será chamado de oferta, e irá corresponder à uma instância da classe do produto.

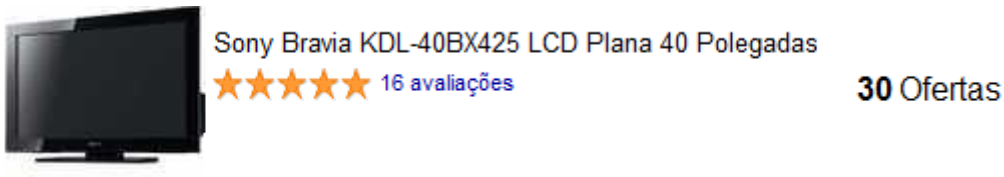

Figura 5.2: Classe de produto Sony Bravia KDL-40BX425

Desta forma, as instâncias (ofertas) são as descrições dos produtos nas lojas virtuais e sabemos que descrições são relativas ao mesmo produto se elas correspondem à mesma classe do produto abstrato no site de comparação de preços.

5.2.2 Domínio Analisado

Nós fizemos *scraping* de todas as categorias disponíveis no web site de comparação de preço, o que inclui diversos tipos de produtos como televisores, monitores, CDs, livros, etc. Pelo domínio extenso que estamos lidando, o

PUC-Rio - Certificação Digital Nº 1012636/CA

Figura 5.3: Instâncias do produto Sony Bravia KDL-40BX425 em diversas lojas

grau de dificuldade da classificação pode variar muito entre as categorias. Para o estudo de caso, selecionamos duas categorias: Televisão e Monitor. Apesar de aparentarem exigir o mesmo esforço de modelagem, são categorias bem distintas quando olhamos mais de perto para as características de suas instâncias.

A categoria de televisores, possui instâncias com descrições desordenadas e com um padrão não muito óbvio, enquanto a categoria de monitor contém instâncias com descrições bem mais regulares e previsíveis.

Com isso iremos necessitar de dois DataSources, um para Televisores e outra para Monitores. O DataSource de Televisores contém 2037 ofertas com 287 produtos, já o de Monitores contém 1592 ofertas com 145 produtos.

O banco de dados gerado contém, para cada classe de produto, uma série de ofertas reais correspondentes a esse produto (instâncias). Refletindo o fato de só ser necessário descobrir a classe de produto de cada oferta. Assim, nosso problema agora é: Dada uma oferta de loja virtual (de TV ou Monitor), qual é a classe de produto relacionado à ela?

5.2.3 Validação Cruzada

Verificar manualmente a acurácia de cada teste inviabilizaria qualquer experimento. Evidentemente, este não foi o procedimento empregado para

PUC-Rio - Certificação Digital Nº 1012636/CA

realizar nossas medições, ao invés disto, utilizamos um artifício chamado de Validação Cruzada, já mencionado na seção 3.2. Para acompanhar nossos progressos ao longo da fase de Engenharia de Atributos, dividimos, para cada categoria, nossos DataSources originais em dois DataSources menores:

1. Conjunto de Treino (TV e Monitor)
2. Conjunto de Validação (TV e Monitor)

É através do conjunto de treino que os classificadores irão aprender as associações entre as classes e as instâncias. Já na fase de validação, para cada instância do conjunto de validação, verificamos qual é o palpite da classe, sugerido pelo classificador, e comparamos com a classe correta que está cadastrada no *dataset*. Desta forma, conseguimos medir a acurácia do nosso modelo.

5.2.4 Divisão do DataSource

Decidimos que aproximadamente 75% das ofertas seriam utilizadas para treino e o restante para a avaliação da qualidade das estratégias. Sendo assim, obtivemos os seguintes números:

Dataset	**Treino**	**Validação**
Monitores	1278	315
Televisores	1575	460

5.2.5 Características Gerais

O *dataset* completo contém 185 categorias, totalizando 194499 classes e 1991849 instâncias - uma média de 10,2 instâncias por classe. Não estamos considerando nesses números classes com apenas uma instância. Abaixo listamos as maiores categorias:

PUC-Rio - Certificação Digital Nº 1012636/CA

5.3 Resultados

Aqui apresentamos os resultados obtidos pelos algoritmos `NaiveBayes`, `VFI`, `NNge` e `J48` com diferentes modelagens.

5.3.1 Monitor

Modelo Inicial

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	22.5397%	296	3829
VFI	93.0159%	344	4921
NNge	77.7778%	45703	8094
J48	90.1587%	6766	141

PUC-Rio - Certificação Digital Nº 1012636/CA

Categoria	**Classes**	**Instancias**	**Média**
Livros	163118	1807784	11,1
Jogos	2906	18160	6,2
Perfume	2488	16333	6,6
Relógio de Pulso	2845	10835	3,8
Tênis	1069	10310	9,6
Cartucho para Impressora	406	4174	10,3
Bonecos e Personagens	948	3736	3,9
Câmera Digital	220	3672	16,7
Circulador / Ventilador	332	3118	9,4
Kit de Som / Alto-Falante para Automóveis	652	3015	4,6
Fone de Ouvido / Headset	503	2875	5,7
Óculos de Sol	733	2841	3,9
Mouse	564	2784	4,9
Cadeira para Auto	247	2698	10,9
Jogos Diversos	579	2664	4,6
Bonecas	591	2525	4,3
Notebook	481	2399	5
Fogão	425	2339	5,5
PC	349	2297	6,6
Bicicleta	310	2290	7,4
Chuteira	283	2200	7,8
HD	349	2158	6,2
Impressora	190	2154	11,3
TV	287	2037	7,1
Aparelho Telefônico	161	1633	10,1
Monitor	145	1592	11
Camisa de Times de Futebol	303	1585	5,2
Refrigerador	201	1585	7,9

Tabela 5.1: Maiores categorias

Modelo inicial com stopwords

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	23.1746%	187	3703
VFI	92.6984%	187	4735
NNge	78.7302%	42984	7407
J48	90.1587%	6219	94

Modelo inicial com novo atributo marca

Classifier	Acertos	Tempo de Aprendizado(ms)	Tempo de avaliação (ms)
NaiveBayes	25.0794%	188	3766
VFI	93.0159%	156	4797
NNge	79.3651%	44860	7937
J48	89.2063%	6312	94

Modelo inicial mais atributo modelo

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	25.0794%	187	3813
VFI	94.9206%	156	4797
NNge	95.2381%	42203	6703
J48	89.8413%	5891	78

Modelo inicial mais atributo tamanho

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação (ms)
NaiveBayes	23.8095%	297	3844
VFI	93.6508%	360	4953
NNge	79.3651%	45422	7968
J48	88.8889%	7156	141

Modelo inicial mais atributo tipo

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	23.8095%	297	3875
VFI	93.0159%	344	4938
NNge	78.7302%	45375	8047
J48	89.8413%	6578	141

PUC-Rio - Certificação Digital Nº 1012636/CA

Modelo inicial mais replicação de instâncias

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	22.5397%	204	4015
VFI	93.0159%	250	4781
NNge	77.7778%	45406	8063
J48	90.1587%	6453	110

Modelo inicial, mais todos os atributos, mais replicação das instâncias, mais stopwords.

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	27.3016%	203	4047
VFI	95.5556%	343	4907
NNge	96.8254%	41937	6438
J48	89.8413%	5469	93

PUC-Rio - Certificação Digital Nº 1012636/CA

5.3.2 TV

Modelo inicial

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	5%	390	11938
VFI	87.3913%	375	13546
NNge	65.4348%	99985	22031
J48	74.5652%	16735	515

Modelo inicial mais stopwords

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	5.2174%	235	11312
VFI	87.1739%	265	13219
NNge	67.1739%	98266	21781
J48	74.3478%	16313	469

Modelo inicial mais atributo marca

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avalição(ms)
NaiveBayes	5.8696%	313	11531
VFI	88.6957%	313	13562
NNge	72.1739%	126625	24562
J48	85.2174%	20422	500

Modelo inicial mais atributo modelo

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	6.087%	297	11469
VFI	90.2174%	312	13579
NNge	90%	123218	21157
J48	84.7826%	20937	484

Modelo inicial mais atributo tamanho

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	6.087%	296	11454
VFI	89.3478%	297	13609
NNge	73.0435%	125141	24406
J48	85.2174%	19891	484

Modelo inicial mais atributo tipo

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	5.4348%	297	11484
VFI	88.6957%	312	13594
NNge	71.087%	125860	24828
J48	85.4348%	20016	500

Modelo inicial mais replicação de instâncias

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	5.4348%	297	11484
VFI	88.6957%	313	13562
NNge	70.2174%	126625	25047
J48	85.4348%	20219	515

Modelo inicial, mais todos os atributos, mais stopwords, mais replicação

Classifier	Acertos	Tempo de aprendizado(ms)	Tempo de avaliação(ms)
NaiveBayes	7.3913%	296	11485
VFI	91.5217%	312	13469
NNge	91.087%	123390	21344
J48	84.7826%	18391	438

PUC-Rio - Certificação Digital Nº 1012636/CA

5.3.3
Exemplos de Resultados da Classificação de TV

A seguir forma relacionados alguns exemplos de televisores (seção 5.3.3) processados por um dos classificadores do *EasyLearn*. A primeira coluna contém as descrições das ofertas nas lojas virtuais e a segunda coluna indica a classe encontrada pelo classificador.

Instância	**Classe**
TV 26"LCD AOC D26W931 c/ Entradas HDMI e USB e Conversor Digital + Antena Interna Sagna Baby Amplificada UHF, VHF e Digital	aoc d26w931 lcd plana 26 polegadas
TV 26"LCD AOC D26W931 c/ Entradas HDMI e USB e Conversor Digital + Limpador de Tela Diamond + Cabo HDMI Diamond Cable 1,5m	aoc d26w931 lcd plana 26 polegadas
TV 32"LCD - D32W931 - (1366 x 768) com Conversor Digital Integrado, 3 Entradas HDMI Entrada USB - AOC	aoc d32w931 lcd plana 32 polegadas
Televisor LCD 32"CCE Stile D3201 HDTV C/ Conversor Digital Integrado, Anti-reflexo, 2 Hdm... cce d32 lcd plana 32 polegadas	cce d32 lcd plana 32 polegadas
TV 14"CCE Cars HDC-142	cce disney hdw 143 crt convencional 14 polegadas
TV 14"CCE Princess HDP-141	cce disney hdw 143 crt convencional 14 polegadas
TV 32"LCD - 3203 HD - 2 Entradas HDMI - H-Buster	h buster hbtv 3203 lcd plana 32 polegadas
TV 21"Tela Plana - 21FJ6RB - c/ Closed Caption - LG	lg flat 21fj6r crt plana 21 polegadas
TV 26"LCD 26PFL3404 (1366 x 768 pixels), 2 Entradas HDMI - Philips	philips 26pfl3404 lcd plana 26 polegadas
Smart TV LED Samsung D5500 com 40"Full HD, Clear Motion Rate 120Hz, Smart Hub e Samsung Apps - UN40D5500RGX	samsung un40d5500 led plana 40 polegadas

PUC-Rio - Certificação Digital Nº 1012636/CA

6
Conclusão

Neste trabalho utilizamos técnicas de aprendizado supervisionado para apresentar uma solução ao problema de classificação de ofertas de produtos do comércio eletrônico, cujo o objetivo é identificar se uma determinada oferta de um produto em uma loja virtual corresponde a um produto já catalogado previamente. Para cumprir o objetivo deste trabalho, construímos um corpus para apoiar as técnicas de aprendizado supervisionado, realizamos a engenharia de atributos para extrair atributos relevantes a partir da descrição da oferta - que é o único atributo disponível à priori, por fim, criamos o *EasyLearn*, um *framework* para apoiar o desenvolvimento de aplicações voltadas ao aprendizado supervisionado. Utilizamos o *framework* desenvolvido para resolver o problema supracitado de forma simples e elegante. Além disto, demonstramos o potencial da infra-estrutura criada demonstrando outras aplicações relacionadas ao aprendizado supervisionado.

Apresentamos o problema de classificação de ofertas de produtos como um estudo de caso do *EasyLearn*.

O *EasyLearn* recebe como entrada um conjunto de arquivos de configuração no formato XML contendo a definição do fluxo de processamento a ser executado, além do DataSource a ser processado - independente do formato. Sua saída é customizável, e pode ser configurada para produzir, por exemplo, relatórios de acurácia da classificação, o próprio DataSource classificado, ou o modelo de classificação já treinado.

A arquitetura do *EasyLearn* foi definida após a análise detalhada dos processos de Classificação, onde identificamos inúmeras atividades em comum entre os três processos estudados (aprendizado, avaliação e classificação). Através desta percepção e tomando as linguagens Orientadas à Objetos como inspiração, criamos um *framework* capaz de comportar os processos de classificação e suas possíveis variações, além de permitir o reaproveitamento das configurações, através da implementação de herança e polimorfismo para os seus arquivos de configuração.

Ilustramos o uso do *framework* criado através de um estudo de caso completo sobre classificação de produtos do comércio eletrônico - incluindo a

PUC-Rio - Certificação Digital Nº 1012636/CA

criação do corpus, engenharia de atributos e análise dos resultados obtidos.

Para solucionar o problema de classificação de produtos, inicialmente, buscamos uma abordagem totalmente baseada no WEKA, sem a utilização do EasyLearn. Entretanto, em função da complexidade do problema a ser resolvido, esbarramos em inúmeros desafios ao utilizar a API do WEKA, entre eles:

- Complexidade da API.
- Necessidade de conhecimento apurado em desenvolvimento JAVA.
- Mensagens de erro pouco intuitivas que não apontavam devidamente para a causa do problema.
- Dificuldade para alterar o fluxo de execução definido inicialmente.

Para lidar com os desafios elencados, o EasyLearn se mostrou eficaz, tornando a solução elaborada simples, modular e escalável para as inúmeras categorias do estudo de caso proposto.

Por fim, como trabalhos futuros, sugerimos:

Novas operações de Aprendizado de Máquina O *Weka* apoia o desenvolvimento de aplicações para aprendizado supervisionado e não-supervisionado, entretanto o *EasyLearn* suporta apenas os classificadores do *Weka*, isto é, apenas as classes de aprendizado supervisionado. O próximo passo a ser dado é suportar um fluxo de execução para englobar operações de agregação, regressão, associação e outros procedimentos suportados pelo *Weka*.

Classificação em tempo real Alguns classificadores podem ser atualizados progressivamente, isto é, sem a necessidade de refazer todo o modelo a cada nova atualização do corpus, estes classificadores no *Weka* herdam da classe `UpdatableClassifier`. A princípio, o *EasyLearn* suportaria um fluxo de conhecimento que utilize classificadores atualizáveis sem a necessidade de modificações na API. No entanto, para termos esta garantia será necessário um estudo adicional nesta direção.

Interface Gráfica Outro caminho a ser trilhado é a criação de uma interface gráfica para facilitar a construção dos *XML* de configuração. O *Weka* já possui uma interface que permite desenhar um fluxo de execução, no entanto, ao salvar o trabalho feito nesta interface, é gerado um arquivo binário em um formato do *Weka*. Além disto, através desta interface, não é possível reaproveitar partes de fluxos de execução definidos em outros momentos do desenvolvimento da solução, o que nos obriga a construir todo o fluxo do início.

PUC-Rio - Certificação Digital Nº 1012636/CA

Criação de camada de abstração Nesta versão do *EasyLearn*, toda a infra-estrutura criada é apoiada sobre o *framework Weka*, o ideal é que o *EasyLearn* possua as suas próprias abstrações e conceitos e utilize o *Weka* como um dos várias *frameworks* de Aprendizado de Máquina. Desta forma, novos *frameworks* poderiam ser integrados ao *EasyLearn*, nos abrindo um leque de possibilidades.

PUC-Rio - Certificação Digital Nº 1012636/CA

Referências Bibliográficas

Alpaydin04. ALPAYDIN, E.. **Introduction to machine learning**. Adaptive computation and machine learning. MIT Press, 2004.

Barnett94. BARNETT, V.; LEWIS, T.. **Outliers in Statistical Data**. Wiley Series in Probability & Statistics. Wiley, Apr. 1994.

Batista03. BATISTA, G. E. A. P. A.; MONARD, M. C.. **An Analysis of Four Missing Data Treatment Methods for Supervised Learning**. Applied Artificial Intelligence, 17:519–533, 2003.

Fellbaum98. FELLBAUM, C.. **WordNet: An Electronic Lexical Database**. Bradford Books, 1998.

Fowler02. FOWLER, M.. **Patterns of Enterprise Application Architecture**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

Gamma95. GAMMA, E.; HELM, R.; JOHNSON, R. ; VLISSIDES, J.. **Design patterns: elements of reusable object-oriented software**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

Hand98. HAND, D.; S.D.JACKA. **Consumer credit and statistics. in statistics in finance**. p. 69–81, 1998. London.

Hastie08. HASTIE, T.; TIBSHIRANI, R. ; FRIEDMAN, J.. **The elements of statistical learning: data mining, inference, and prediction**. Springer series in statistics. Springer, 2008.

Hodge04. HODGE, V.; AUSTIN, J.. **A survey of outlier detection methodologies**. Artif. Intell. Rev., 22(2):85–126, Oct. 2004.

Indurkhya10. INDURKHYA, N.; DAMERAU, F. J.. **Handbook of Natural Language Processing**. Chapman & Hall/CRC, 2nd edition, 2010.

Jaakkola98. JAAKKOLA, T.; HAUSSLER, D.. **Exploiting generative models in discriminative classifiers**. In: IN ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 11, p. 487–493. MIT Press, 1998.

PUC-Rio - Certificação Digital Nº 1012636/CA

Kegelmeyer02. CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O. ; KEGELMEYER, W. P.. **Smote: Synthetic minority over-sampling technique**. Journal of Artificial Intelligence Research, 16:321–357, 2002.

Kotsiantis07. KOTSIANTIS, S. B.. **Supervised machine learning: A review of classification techniques. informatica 31:249?268**, 2007.

Kukich92. KUKICH, K.. **Techniques for automatically correcting words in text**. ACM Comput. Surv., 24(4):377–439, Dec. 1992.

Larman04. LARMAN, C.. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)**. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

Lewis98. LEWIS, D. D.. **Naive (bayes) at forty: The independence assumption in information retrieval**. In: ?, p. 4–15. Springer Verlag, 1998.

Manning99. MANNING, C. D.; SCHÜTZE, H.. **Foundations of statistical natural language processing**. MIT Press, Cambridge, MA, USA, 1999.

Maron61. MARON, M. E.. **Automatic indexing: An experimental inquiry**. J. ACM, 8:404–417, July 1961.

Markovitch02. MARKOVITCH, S.; ROSENSTEIN, D.. **Feature generation using general constructor functions**. Machine Learning, 49:59–98, 2002. 10.1023/A:1014046307775.

Michie94. MICHIE, D.; SPIEGELHALTER, D. ; TAYLOR, C.. **Machine learning, neural and statistical classification**. Artificial intelligence. Ellis Horwood, 1994.

Mitchell97. MITCHELL, T.. **Machine Learning**. McGraw-Hill series in computer science. McGraw-Hill, 1997.

Motoda01. LIU, H.; MOTODA, H.. **Instance Selection and Construction for Data Mining**. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

Quinlan93. QUINLAN, J.. **C4.5: programs for machine learning**. Morgan Kaufmann series in machine learning. Morgan Kaufmann Publishers, 1993.

Simeon08. SIMEON, M.; HILDERMAN, R.. **Categorical proportional difference: A feature selection method for text categorization**. In:

PUC-Rio - Certificação Digital Nº 1012636/CA

Roddick, J. F.; Li, J.; Christen, P. ; Kennedy, P. J., editors, SEVENTH AUSTRALASIAN DATA MINING CONFERENCE (AUSDM 2008), volumen 87 de **CRPIT**, p. 201–208, Glenelg, South Australia, 2008. ACS.

Smola08. SMOLA, A. J.; VISHWANATHAN, S.. **Introduction to Machine Learning**. Press Syndicate of the University of Cambridge, 2008.

Sproat01. SPROAT, R.; BLACK, A. W.; CHEN, S.; KUMAR, S.; OSTENDORF, M. ; RICHARDS, C.. **Normalization of non-standard words**. Computer Speech and Language, 15(3):1–40, 2001.

Teller00. **Review of "speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition"by daniel jurafsky and james h. martin. prentice hall 2000.** Comput. Linguist., 26(4):638–641, Dec. 2000. Reviewer-Teller, Virginia.

Thor. THOR, A.. **Toward an adaptive string similarity measure for matching product offers**.

Tompkins98. TOMPKINS, J.; SMITH, J.. **Warehouse Management Handbook**. Tompkins Press, 1998.

Unicode11. CONSORTIUM, T. U.. **Unicode normalization forms**. http://unicode.org/reports/tr15/, 08 2012. 31.

WebShoppers11. EBIT EMPRESA. **Webshoppers - 24ª edição**. `http://www.webshoppers.com.br/webshoppers/WebShoppers24.pdf`, Agosto 2011.

Weka08. PROJECT, W. M. L.. **Weka**. http://www.cs.waikato.ac.nz/ ml/weka.

Woods93. WOODS, K. S.; DOSS, C. C.; BOWYER, K. W.; SOLKA, J. L.; PRIEBE, C. E. ; KEGELMEYER, W. P.. **Comparative evaluation of pattern recognition techniques for detection of microcalcifications in mammography**. International Journal of Pattern Recognition and Artificial Intelligence, 1993.

Yu2004. YU, L.; LIU, H.. **Efficient feature selection via analysis of relevance and redundancy**. J. Mach. Learn. Res., 5:1205–1224, Dec. 2004.

eSchool11. SCHOOL, E.. **Informações do comércio eletrônico no brasil**. http://www.camara-e.net/2011/09/01/comercio-eletronico-nao-encontra-profissionais-qualificados/, 09 2011. 01.

PUC-Rio - Certificação Digital Nº 1012636/CA

A
Classes do EasyLearn

A.1
Pacote Matcher

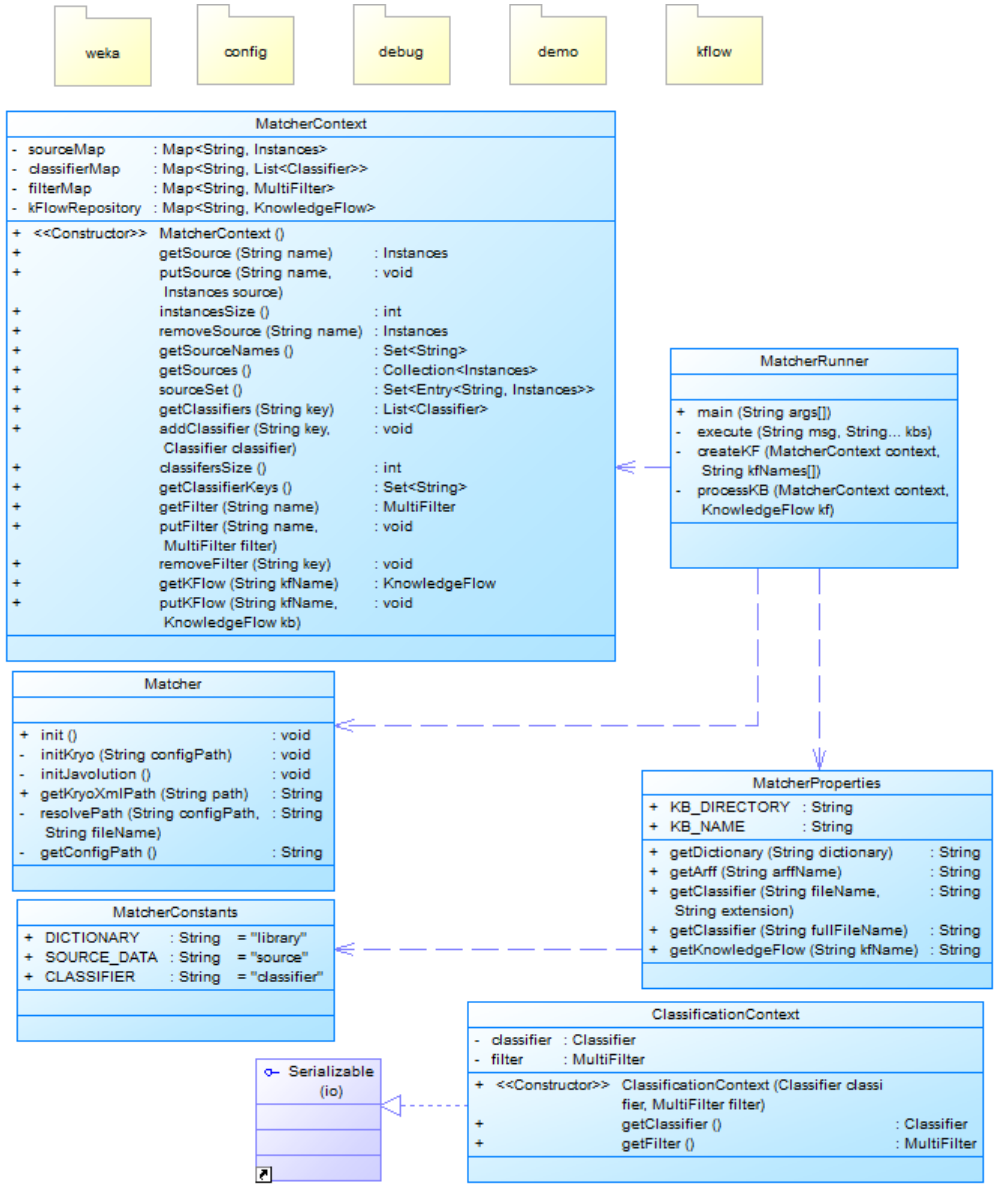

Figura A.1: Diagrama das classes do pacote Matcher

PUC-Rio - Certificação Digital Nº 1012636/CA

MatcherRunner Este é o ponto de entrada da API. O `MatcherRunner` realiza a carga de todos os arquivos de configuração necessários, de forma ordenada, cria um `MatcherContext` e chama o `MatcherFlowController` (seção A.3) para iniciar fluxo escolhido (Avaliação, Aprendizado, Classificação, ou qualquer outro fluxo que siga os padrões definidos).

Matcher Ponto central de configuração da API, carregando todas as propriedades do framework, inicializando (`Matcher.init()`) serviços básicos de Serialização, Logging e qualquer outra biblioteca que necessite de configurações adicionais.

MatcherConstants Contém as constantes utilizadas no *framework*. Evita que as constantes de configuração fiquem espalhadas por todo o código-fonte.

MatcherContext Implementa o padrão de projeto *Repository* definido em (Fowler02). Conforme o `MatcherFlowController` executa cada uma de suas etapas, cada componente pode alterar o estado do `MatcherContext`, modificando o contexto corrente. O `MatcherContext` possui uma relação com todos os `InstanceSource` carregados pela execução do processo corrente, além de todos os classificadores utilizados. Através do `MatcherContext` é possível realizar qualquer operação sobre os conjuntos de dados abertos. Em suma, este objeto é responsável por guardar a saída da etapa anterior, que pode ser utilizada como entrada para a etapa seguinte.

MatcherProperties Provê e carrega todas as propriedades variáveis do *framework*, tais como configuração da API de *logging*, de serialização e de classificação. Além de manter as propriedades referentes ao diretório da *KnowledgeBase* a ser utilizada.

ClassificationContext Ao final do processo de aprendizado, é natural persistirmos o modelo gerado para ser utilizado futuramente, sobre um outro conjunto de dados. Para que o classificador seja útil ao ser carregado novamente (desserializado) precisamos realizar as mesmas operações (pré-processamento) que realizamos sobre o novo conjunto de dados, caso contrário, é possível que tenhamos um resultado inesperado, ou alguma incompatibilidade com os modelos gerados. Para solucionar este problema, temos que persistir não apenas o classificador, mas também, todo o seu contexto de classificação (`ClassificationContext`), isto é, os filtros aplicados sobre o conjunto de treino - já na ordem correta de execução.

PUC-Rio - Certificação Digital Nº 1012636/CA

A.2
Pacote Matcher.Config

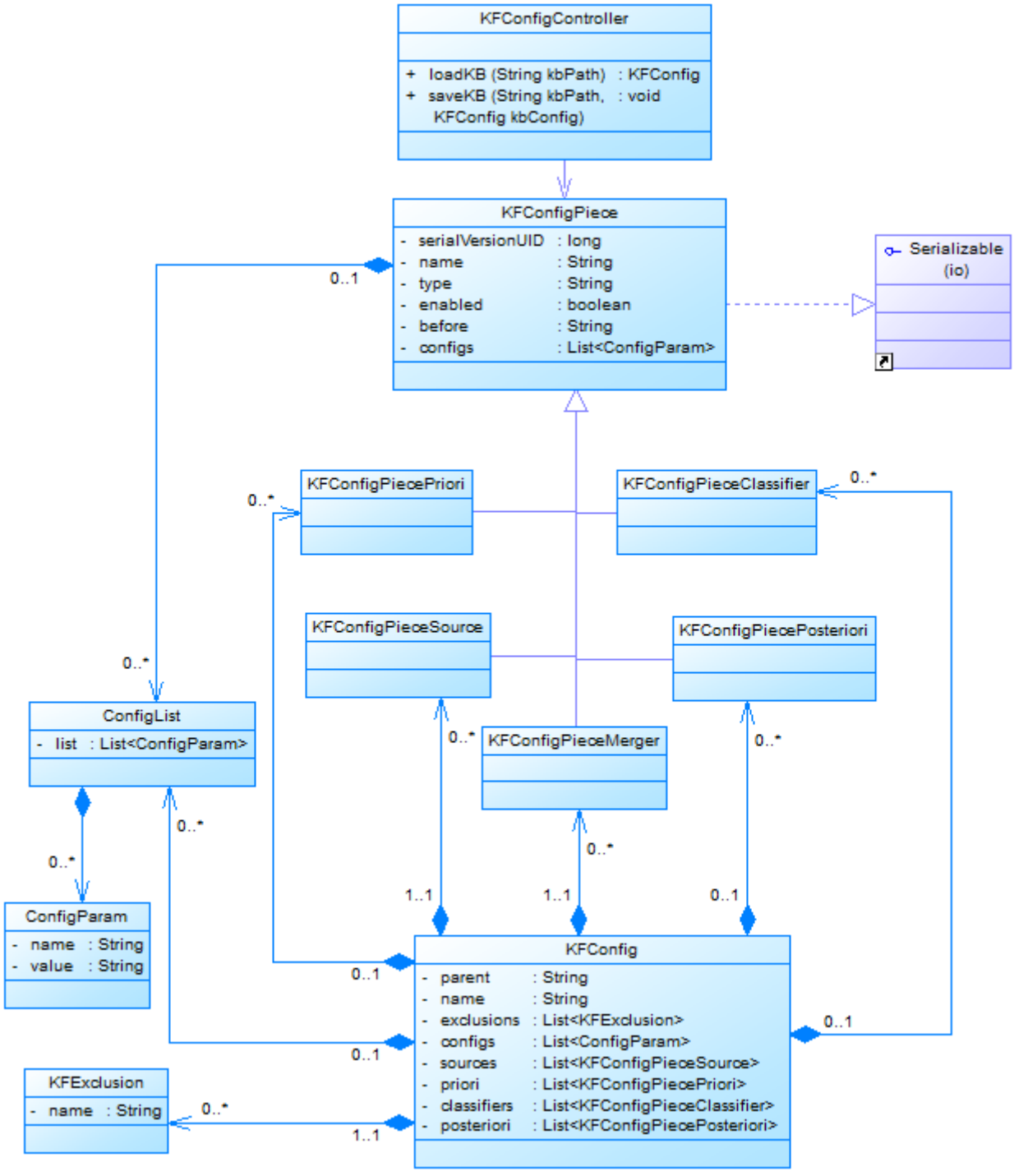

Figura A.2: Diagrama das classes do pacote Matcher.Config

KFConfigController Classe de entrada do módulo de configuração do *EasyLearn*, através do `KFConfigController` é possível, dado o caminho do arquivo de configuração do *KnowledgeFlow*, carregar o `KFConfig` correspondente. Para isto, o `KFConfigController` faz o *parsing* do arquivo de configuração

PUC-Rio - Certificação Digital Nº 1012636/CA

informado como parâmetro do método `loadKB`. Esta classe também possibilita seguir pelo caminho inverso: dado um destino e um `KFConfig`, o `KFConfigController` persiste o conteúdo do `KFConfig` informado, no formato definido pelo *XSD* da seção 4.4.

KFConfig O `KFConfig`, ou *KnowledgeFlow Config* é uma representação de objeto do XML de configuração do *KnowledgeFlow*. Através dos métodos desta classe é possível abstrair todas as dificuldades de se trabalhar diretamente com o arquivo XML. Uma vez que o XML de configuração seja processado pelo `KFConfigController` toda a informação armazenada no XML é carregada em um objeto desta classe.

KFConfigPiece Cada pedaço do XML de configuração especificado na seção 4.4 herda desta classe. Esta classe fornece a estrutura básica de cada segmento do XML. Ao estabelecermos uma super classe comum a todas as classes de configuração do *EasyLearn* definimos um padrão que simplifica a escrita e o entendimento do XML de configuração. Cada subclasse de `KFConfigPiece` é mapeada em um objeto `<include>` do XML (vide XSD na seção 4.4).

KFConfigPieceSource Corresponde ao bloco `<sources>` do XML de configuração. Mantém as configurações de todos os *DataSources* informados no *KnowledgeFlow*. Ver seção A.4 para mais informações.

KFConfigPiecePriori Corresponde ao bloco `<priori>` do XML de configuração. Mantém as configurações de todos os pré-processadores informados no *KnowledgeFlow*. Ver seção A.5 para mais informações.

KFConfigPieceMerger Corresponde ao bloco `<merger>` do XML de configuração. Responsável por guardar a configuração do agregador de filtros a ser utilizado pelo *framework*. O conceito de agregador de filtros é explicado na seção A.7.

KFConfigPieceClassifier Corresponde ao bloco `<classifiers>` do XML de configuração. Mantém as configurações de todos os classificadores informados no *KnowledgeFlow*. Ver seção A.6 para mais informações.

KFConfigPiecePosteriori Corresponde ao bloco `<posteriori>` do XML de configuração. Mantém as configurações de todos os pós-processadores informados no *KnowledgeFlow*. Ver seção A.8 para mais informações.

KFExclusion Corresponde ao bloco `<exclusions>` do XML de configuração. Mantém as configurações de todos os `KFConfigPiece` definidos no XML

PUC-Rio - Certificação Digital Nº 1012636/CA

de configuração referenciado pelo atributo `parent` do *KnowledgeFlow* que devem ser desconsiderados no XML de configuração filho.

ConfigParam A classe `KFConfigPiece`, consequentemente, todas as classes da sua hierarquia, possuem a capacidade de armazenarem parâmetros de configuração (`ConfigParam`) do tipo *chave-valor*. Através destes parâmetros é possível alcançar um nível maior de flexibilidade para o *framework*. Estes parâmetros são consumidos mais tarde, durante a fase de execução do *KnowledgeFlow*.

ConfigList Lista de `ConfigParam` que fornece um conjunto de métodos que simplificam o acesso aos parâmetros de configuração. Todos os parâmetros de configuração vêm do XML como *String*. Com os métodos da `ConfigList` podemos recuperar os parâmetros como valores inteiros, booleanos, datas, classes, sem que seja necessário fazer *parse* da *String*, já que isto é feito de forma transparente por esta classe.

PUC-Rio - Certificação Digital Nº 1012636/CA

A.3
Pacote Matcher.KFlow

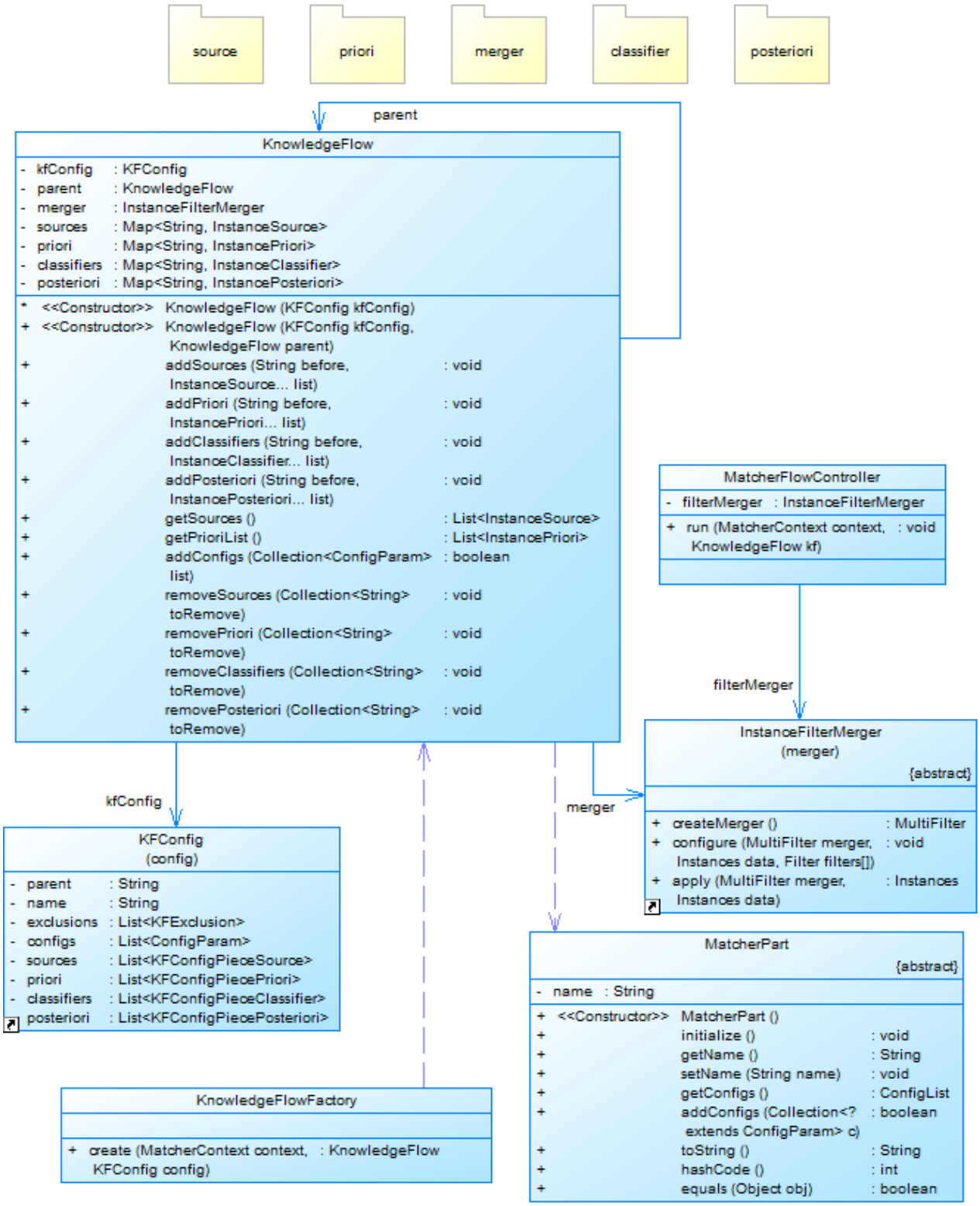

Figura A.3: Diagrama das classes do pacote Matcher.KFlow

PUC-Rio - Certificação Digital Nº 1012636/CA

MatcherPart Classe abstrata responsável por fornecer os métodos básicos para todos os objetos que fazem parte da classe `KnowledgeFlow`. Mantém a lista de parâmetros de configuração (`ConfigList` carregada no XML de configuração e um atributo `name`).

KnowledgeFlow . O `KnowledgeFlow` é construído através de um objeto do tipo `KFConfig` já carregado com informações do XML de configuração, e, opcionalmente, por um `KFConfig` representando o conjunto de configurações do XML raiz (definido pelo atributo *parent*). Nesta classe é guardado todos os `MatcherPart` que serão executados pelo `MatcherFlowController`. Os construtores desta classe são restritos ao pacote Matcher.KFlow (ver `KnowledgeFlowFactory`).

KnowledgeFlowFactory Implementa o padrão de projeto *Factory* definido em (Gamma95). A criação de objetos `KnowledgeFlow` deve ser intermediada por esta *classe*. Esta classe também é responsável por estabelecer a ordem de junção dos `MatcherPart` de acordo com o atributo `before` do `<include>` (`<include before='MatcherPartName' ...>`), mantendo a sequência definida pelo usuário da API no momento da criação do XML de configuração, além de habilitar ou desabilitar um `MatcherPart` pelo atributo `enabled` (`<include ... enabled='false'>`).

MatcherFlowController Define um fluxo padrão de execução para cada `KnowledgeFlow`. Possui um único método: `run(MatcherContext, KnowledgeFlow)`. Além do `KnowledgeFlow`, o método `run(...)` também recebe um `MatcherContext` que acumula as operações realizadas por cada `MatcherPart` definida no `KnowledgeFlow`. Caso seja necessário modificar o controlador de fluxo a ser utilizado, é possível definir um controlador alternativo através do XML de configuração, utilizando o atributo `flow`.

PUC-Rio - Certificação Digital Nº 1012636/CA

A.4
Pacote Matcher.KFlow.Source

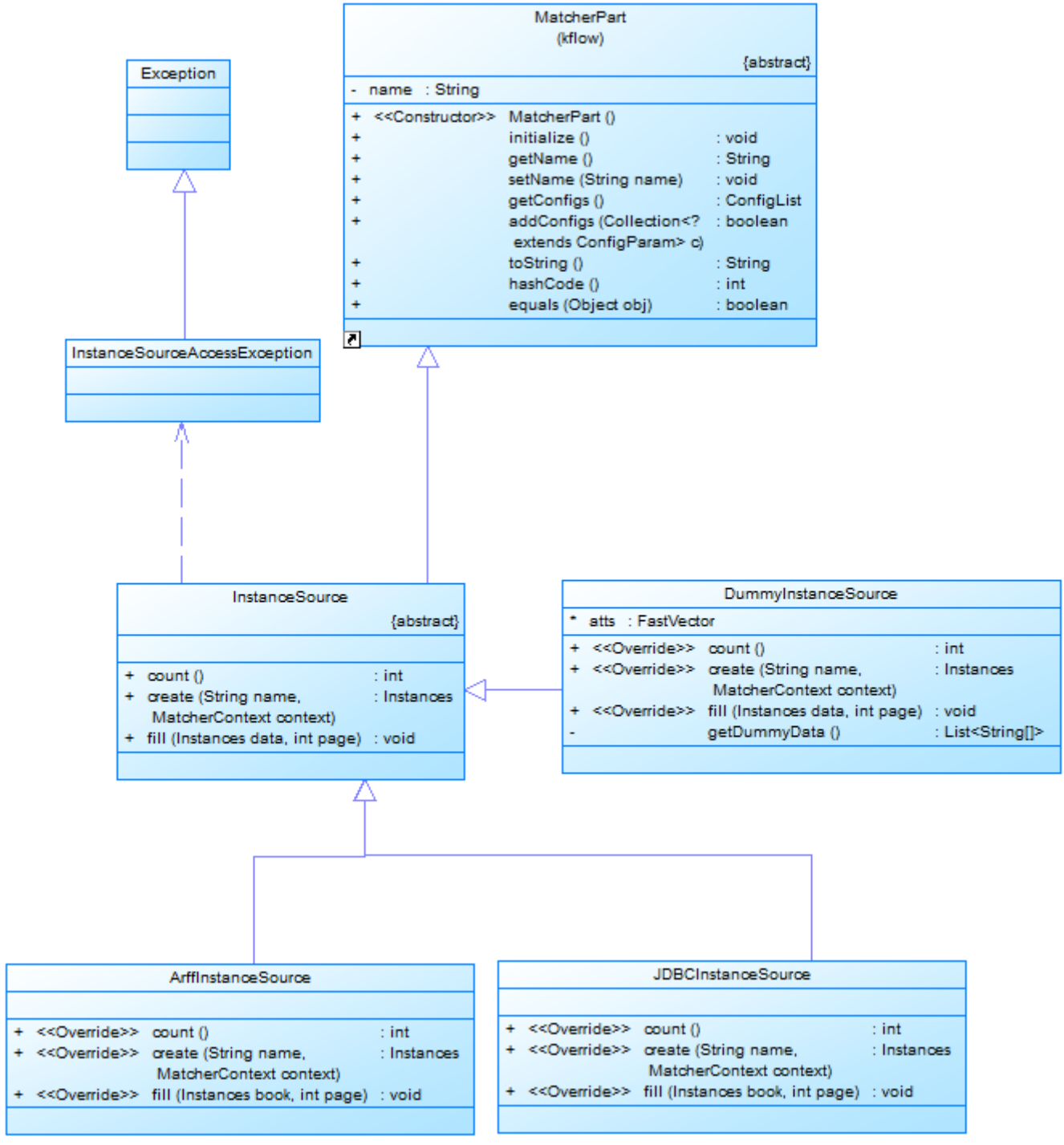

Figura A.4: Diagrama das classes do pacote Matcher.KFlow.Source

InstanceSource Classe abstrata que estabelece os métodos básicos para a criação de componentes para abertura e carregamento de *DataSource* utilizado no *EasyLearn*. Os seguintes métodos foram definidos nesta classe:

`create(...)`: cria e adiciona o DataSource no `MatcherContext` recebido

PUC-Rio - Certificação Digital Nº 1012636/CA

por parâmetro.
`fill(...)`: carrega os dados referentes à página recebida como parâmetro.
`count()`: retorna o total de páginas do *DataSource* a ser carregado.

ArffInstanceSource Herda de `InstanceSource` e carrega o *DataSource* de um arquivo *arff*.
Requer os seguintes atributos:
`arff:String` - caminho relativo ao diretório *library* do *KnowledgeBase* do arquivo *ARFF* (padrão de arquivo de entrada do *WEKA*) a ser carregado.
`classIndex:Integer` - índice do atributo de classe do *DataSource*.

JDBCInstanceSource Herda de `InstanceSource` e carrega o *DataSource* de um banco de dados qualquer através da API *JDBC* do Java.
Requer os seguintes atributos:
`connString:String` - String de conexão ao banco de dados.
`sql:String` - Consulta SQL utilizada para extrair os dados.

DummyInstanceSource Herda de `InstanceSource` e permite construir um conjunto de dados aleatórios para testes do *framework*.

InstanceSourceAccessException Cada método de acesso à camada de dados pode lançar uma exceção do tipo `InstanceSourceAccessException`, seja ao tentar ler de uma base de dados indisponível, ou um arquivo que esteja inacessível, ou qualquer outro motivo.

PUC-Rio - Certificação Digital Nº 1012636/CA

A.5
Pacote Matcher.KFlow.Priori

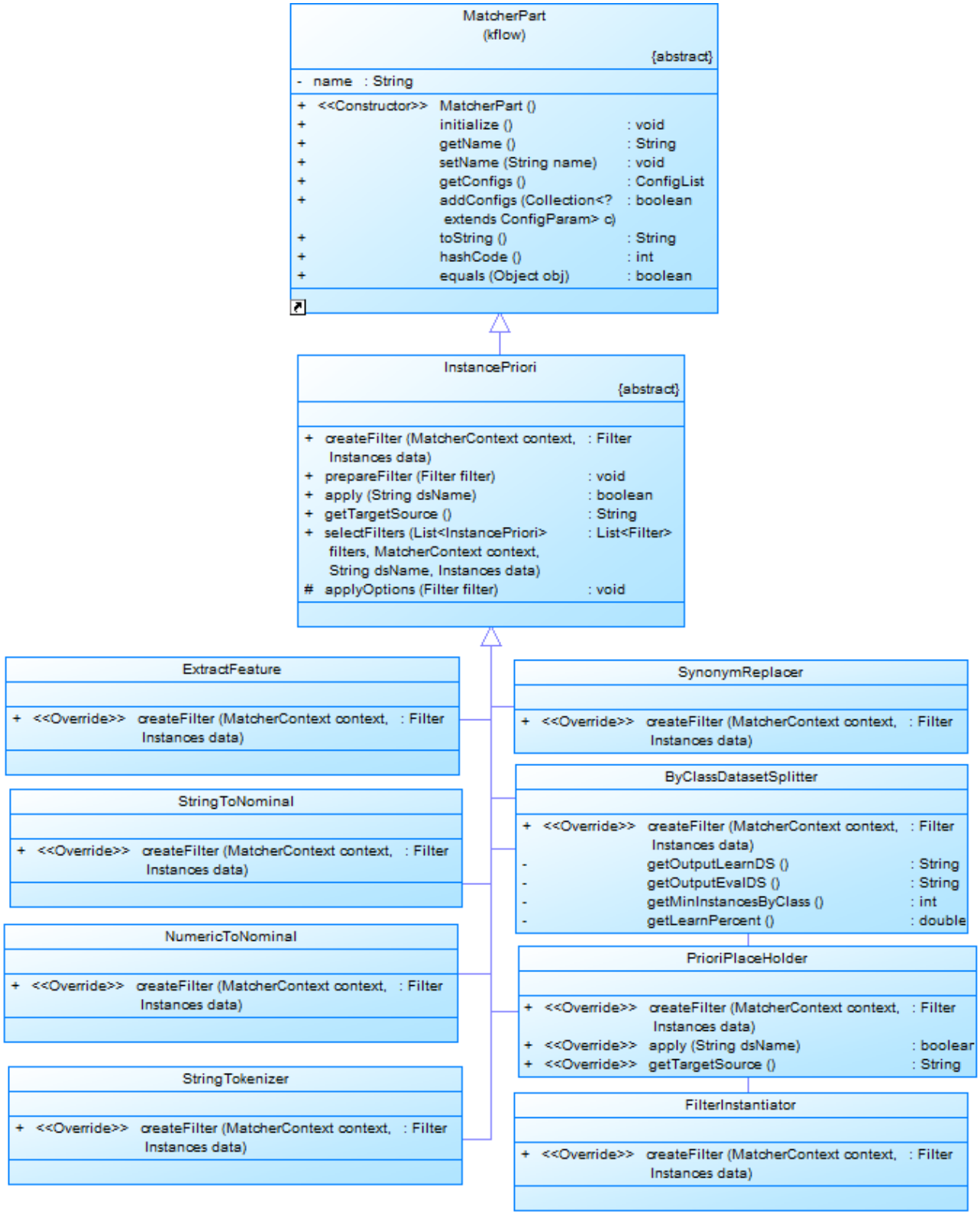

Figura A.5: Diagrama das classes do pacote Matcher.KFlow.Priori

PUC-Rio - Certificação Digital Nº 1012636/CA

InstancePriori Classe abstrata que estabelece os métodos básicos para a criação de componentes para pré-processamento do *DataSource* utilizado no *EasyLearn*. Os seguintes métodos foram definidos nesta classe:

`createFilter(...)`: implementa o padrão de projeto *TemplateMethod* de (Gamma95), este método deve configurar e retornar uma implementação da classe `Filter` do *framework WEKA* que será utilizado para pré-processar os *DataSources*.

`prepareFilter(...)`: configura o filtro criado de acordo com o parâmetro facultativo `options`.

`apply(...)`: retorna `true` caso o filtro deva ser aplicado para o *DataSource* corrente.

Parâmetro obrigatório:
`targetSource:String` - nome do *DataSource* a ser aplicado o filtro.

Parâmetro facultativo:
`options:String[]` - lista de parâmetros de configuração do filtro, separados por vírgula.

FilterInstantiator Herda de `InstancePriori`. Esta classe possibilita ao desenvolvedor carregar e instanciar filtros dinamicamente, através do uso do pacote de *reflection* do Java.
Parâmetro obrigatório:
`type:Class` - nome completo da classe a ser carregada. A classe a ser instanciada deve pertencer à hierarquia da classe `Filter` do WEKA.

StringTokenizer Herda de `InstancePriori`. Converte um atributo *Text* do *DataSource* em uma série de atributos, cada qual, representando uma palavra. O processo de separação das palavras depende do `Tokenizer` a ser utilizado, e também pode ser configurado através do parâmetro `options`. Este filtro deve ser aplicado para a construção da *Bag of Words* explicada na seção 2.4. Cada atributo criado será do tipo *Numeric* do ARFF.

NumericToNominal Herda de `InstancePriori`. Após aplicar o filtro `StringTokenizer`, recomenda-se utilizar o filtro `NumericToNominal`, pois os atributos gerados após o processo de separação das palavras serão do tipo *Numeric*, e, nem sempre serão suportados pelo classificador utilizado, dependendo do caso. Este filtro transforma os atributos numéricos especificados, em atributos do tipo *Nominal* que possuem um conjunto bem definido de valores.

PUC-Rio - Certificação Digital Nº 1012636/CA

StringToNominal Herda de `InstancePriori`. Converte atributos do tipo *Text* do padrão *ARFF*, em atributos do tipo *Nominal*. Esta transformação é imprescindível em alguns casos, especialmente porque a maioria dos classificadores não suportam atributos do tipo *Text*.

SynonymReplacer Herda de `InstancePriori`. Substitui sinônimos por palavras correspondentes visando diminuir a dimensão do *DataSource*, tornando a classificação mais eficiente e simplificando o modelo gerado. Utiliza-se de um dicionário de sinônimos e correspondência de palavras ou termos para realizar esta operação. O dicionário a ser informado deve seguir algumas regras simples. Exemplo:

```
ajudar == auxiliar|socorrer
codigo = \bcod(\d+)\b|\bcod\s(\d+)\b
```

Podemos ver pelo exemplo acima duas possíveis entradas para o dicionário de sinônimos. A primeira linha informa ao filtro que devemos substituir as palavras `auxiliar` e `socorrer` pela palavra `ajudar` (utilizando-se `"=="` representa uma substituição literal). A linha de baixo, informa que devemos realizar a substituição de uma expressão regular (cod seguido de algum número) pela palavra `codigo` (ao utilizar `"="` podemos informar uma expressão regular).

Parâmetros obrigatórios:
`dictionary:String` - caminho relativo ao diretório *library* do *KnowledgeBase* do arquivo de dicionário a ser carregado.

`targetIndex:Integer` - índice do atributo alvo a ser aplicado o filtro.

ExtractFeature Herda de `InstancePriori`. Extrai um novo atributo *Nominal* para o *DataSource*, tomando como base algum atributo do tipo *Text* do ARFF. Este pré-processador utiliza-se de um dicionário de termos, e faz uma busca em cada palavra do atributo alvo, caso haja alguma palavra que esteja contida no dicionário de termos, esta palavra vira valor do atributo a ser criado.

Parâmetros obrigatórios:
`dictionary:String` - caminho relativo ao diretório *library* do *KnowledgeBase* do arquivo de dicionário a ser carregado.

`newFeatureName:String` - nome do novo atributo a ser criado no *DataSource*.

`targetFeatureName:String` - nome atributo alvo a ser aplicado o filtro.

PUC-Rio - Certificação Digital Nº 1012636/CA

ByClassDatasetSplitter Herda de `InstancePriori`. Auxilia na quebra do *DataSource* definido pelo atributo `targetSource` em *DataSource* de treino e *DataSource* de avaliação, seguindo algumas regras básicas.

Parâmetros obrigatórios:
`outLearnSource:String` - nome do *DataSource* de treino a ser criado pelo filtro.

`outEvalSource:String` - nome do *DataSource* de avaliação a ser criado pelo filtro.

`minInstancesByClass:Integer` - quantidade mínima de instâncias que cada classe deve possuir para ser incluída nos *DataSources* de saída. Classes que não atinjam esta quantidade de instâncias serão eliminadas no *output* gerado.

`learnPercent:Double` - percentual aproximado do tamanho do *DataSource* de treino em relação ao *DataSource* de avaliação.

PrioriPlaceHolder Esta classe não possui implementação e não interfere no *DataSource*. Ela é utilizada apenas para facilitar a criação de arquivos de configuração que possuem uma hierarquia mais complexa. Através do *place holder* é possível "reservar"um espaço no arquivo de configuração do XML pai, facilitando a inserção de outros elementos no XML filho na posição correta.

PUC-Rio - Certificação Digital Nº 1012636/CA

A.6
Pacote Matcher.KFlow.Classifier

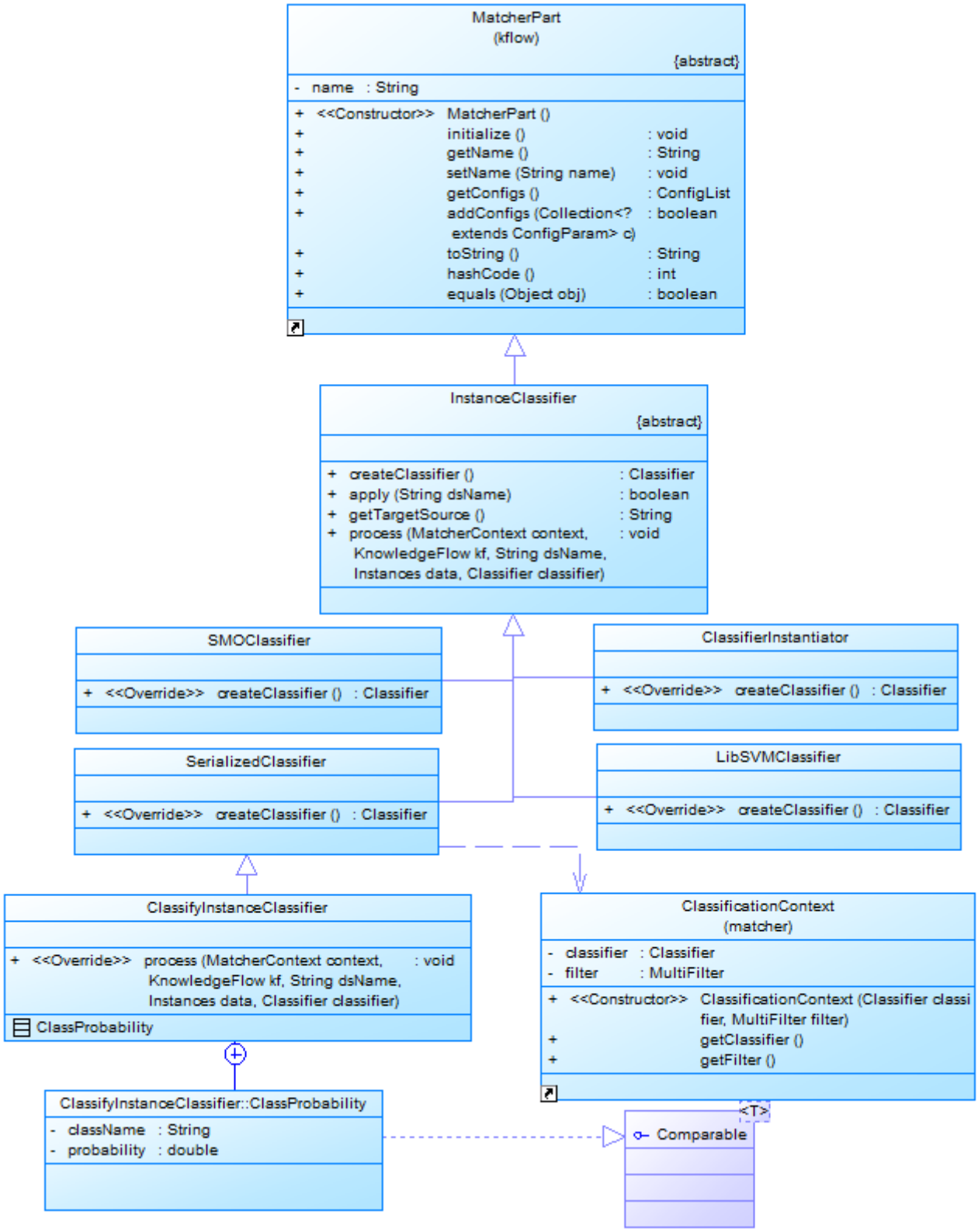

Figura A.6: Diagrama das classes do pacote Matcher.KFlow.Classifier

PUC-Rio - Certificação Digital Nº 1012636/CA

InstanceClassifier Classe abstrata que estabelece os métodos básicos para a criação de componentes para classificação do *DataSource* utilizado no *EasyLearn*. Os seguintes métodos foram definidos nesta classe:

`createClassifier()`: implementa o padrão de projeto TemplateMethod de (Gamma95), este método deve configurar e retornar uma implementação da classe `Classifier` do *framework* WEKA que será utilizado para classificar os *DataSources*.

`apply(...)`: retorna `true` caso o classificador deva ser aplicado para o *DataSource* corrente.

`process(...)`: realiza a classificação sobre o *DataSource* alvo, o resultado da classificação deve ser adicionado dentro do `MatcherContext` para acesso futuro por outras etapas do fluxo de execução do `KnowledgeFlow`.

Parâmetro obrigatório:
`targetSource:String` - nome do *DataSource* a ser aplicado o classificador.

ClassifierInstantiator Herda de `InstanceClassifier`. Esta classe possibilita ao desenvolvedor carregar e instanciar classificadores dinamicamente, através do uso do pacote de *reflection* do Java.
Parâmetro obrigatório:
`type:Class` - nome completo da classe a ser carregada. A classe a ser instanciada deve pertencer à hierarquia da classe `Classifier` do WEKA.

SMOClassifier Herda de `InstanceClassifier`. Carrega e configura o algoritmo de classificação `SMO` que é a implementação do classificador `SVM`.
Parâmetro facultativo:
`c:Double` - parâmetro de complexidade do algoritmo `SVM`.

LibSVMClassifier Herda de `InstanceClassifier`. Carrega e configura o algoritmo de classificação `SVM` utilizando uma implementação mais eficiente do framework `LibSVM`.
Parâmetro facultativo:
`c:Double` - parâmetro de complexidade do algoritmo `SVM`.

SerializedClassifier Herda de `InstanceClassifier`. Carrega um `ClassificationContext` serializado do diretório `classifier` da `KnowledgeBase`.

Parâmetro obrigatório:
`fileName:String` - caminho relativo ao diretório *classifier* do `KnowledgeBase` do arquivo do classificador serializado a ser carregado.

PUC-Rio - Certificação Digital Nº 1012636/CA

ClassifyInstanceClassifier Herda de `SerializedClassifier`. Este classificador é utilizado ao final do processo de classificação 3.4 para processar o *DataSource* contendo os dados reais. Ao final da execução deste classificador, o `MatcherContext` é atualizado para guardar o resultado da classificação. Para cada instância classificada é criado uma lista de objetos do tipo `ClassProbability` que guarda o palpite da classe relativa àquela instância, e um `double` representando a certeza daquele palpite (probabilidade).

PUC-Rio - Certificação Digital Nº 1012636/CA

A.7
Pacote Matcher.KFlow.Merger

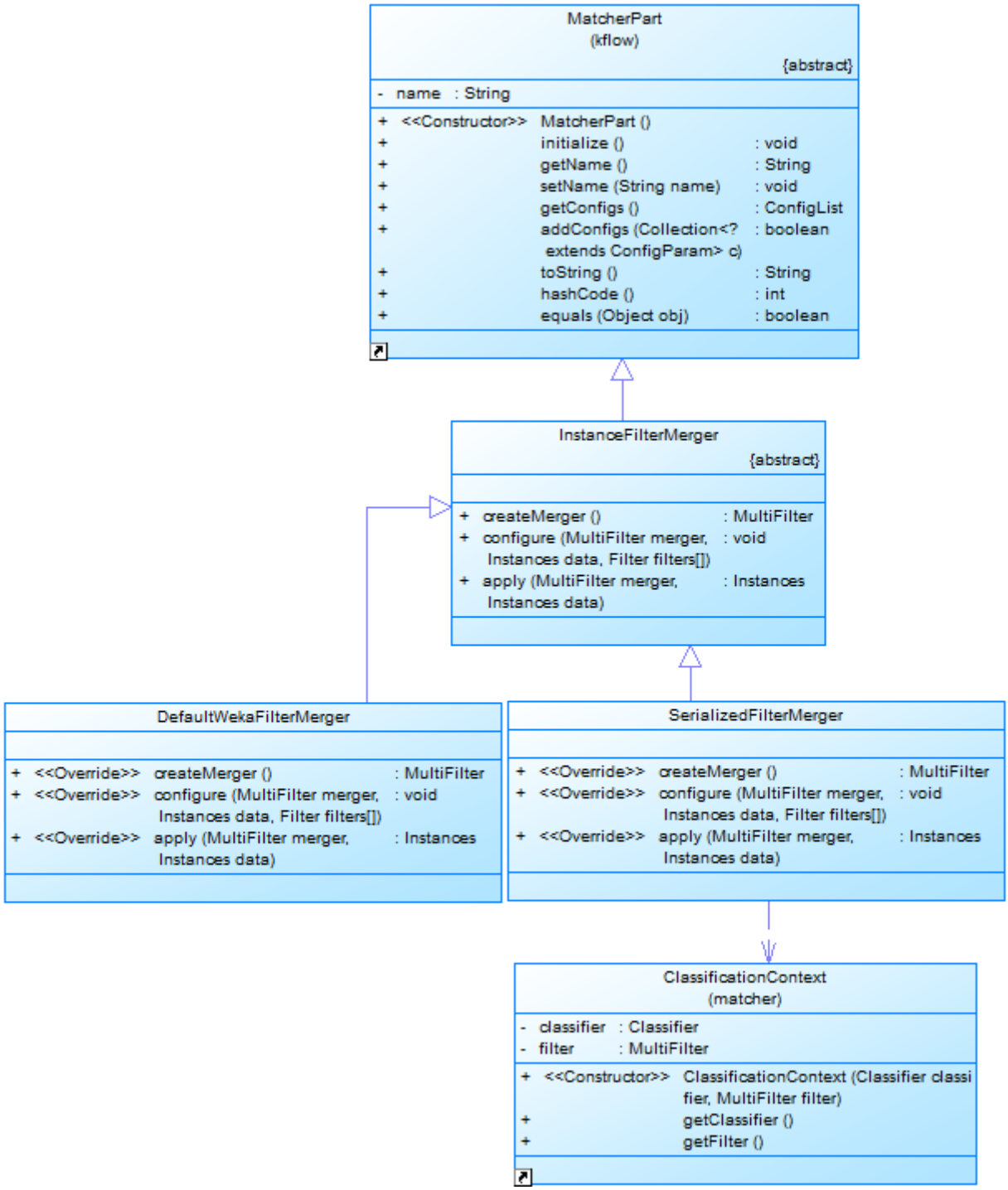

Figura A.7: Diagrama das classes do pacote Matcher.KFlow.Merger

InstanceFilterMerger Classe abstrata que estabelece os métodos básicos para a criação de componentes para pré-processamento em lote do *DataSource* utilizado no *EasyLearn*. Os seguintes métodos foram definidos nesta classe:

PUC-Rio - Certificação Digital Nº 1012636/CA

`createMerger()`: implementa o padrão de projeto `TemplateMethod` de (Gamma95), este método deve configurar e retornar uma implementação da classe `MultiFilter` do *framework WEKA* que será utilizado para pré-processar de forma mais eficiente os *DataSources*.

`apply(...)`: recebe um *dataset* bruto e aplica o conjunto de pré-processadores, retornando um novo *dataset* pronto para a classificação.

DefaultWekaFilterMerger Herda de `InstanceFilterMerger`. Implementação padrão de *FilterMerger* utilizando o `MultiFilter` do WEKA.

SerializedFilterMerger Herda de `InstanceFilterMerger`. Carrega um `ClassificationContext` serializado do diretório `classifier` da *KnowledgeBase* e utiliza o `FilterMerger` do `ClassificationContext` para realizar o pré-processamento. Desta forma, garantimos que o mesmo conjunto de operações aplicados no *dataset* durante o processo de Aprendizado também será aplicado durante a fase de classificação - assegurando que os *datasets* vão possuir a mesma estrutura ao final do processamento e evitando inconsistências e incompatibilidades entre eles.

PUC-Rio - Certificação Digital Nº 1012636/CA

A.8
Pacote Matcher.KFlow.Posteriori

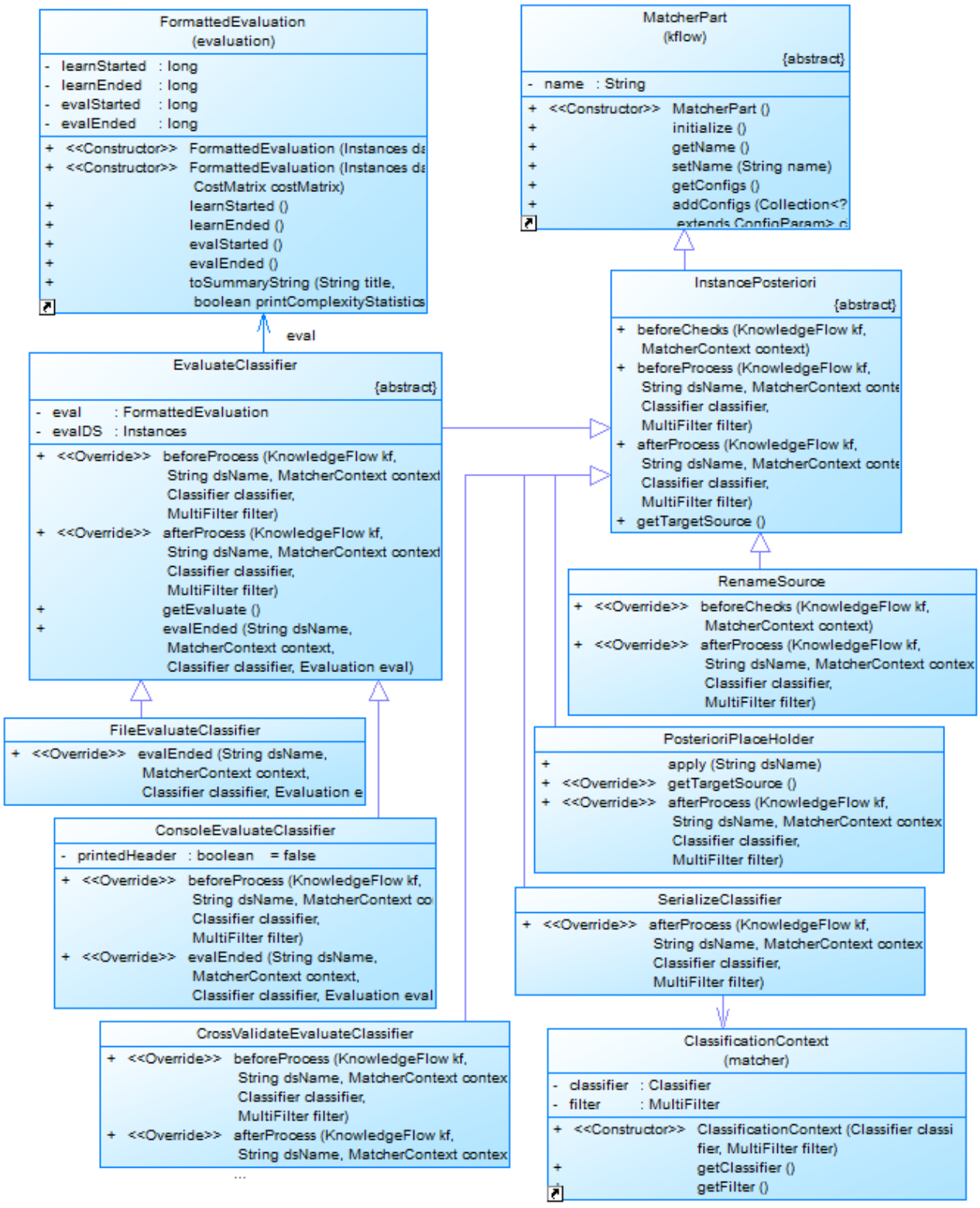

Figura A.8: Diagrama das classes do pacote Matcher.KFlow.Posteriori

PUC-Rio - Certificação Digital Nº 1012636/CA

InstancePosteriori Classe abstrata que estabelece os métodos básicos para a criação de componentes para pós-processamento do *DataSource* utilizado no *EasyLearn*. Os seguintes métodos foram definidos nesta classe:

`beforeChecks(...)`: evento acionado pelo controlador de fluxo antes do processo de validação do `MatcherContext`.

`beforeProcess(...)`: evento acionado pelo controlador de fluxo antes do processo de classificação.

`afterProcess(...)`: evento acionado pelo controlador de fluxo após o processo de classificação.

Parâmetro obrigatório:
`targetSource:String` - nome do *DataSource* a ser aplicado o pós-processador.

EvaluateClassifier Herda de `InstancePosteriori` e utiliza a classe `Evaluation` do WEKA para realizar as avaliações de precisão e tempo gasto para treinamento e classificação. Parâmetro obrigatório:
`evalSource:String` - nome do *DataSource* de avaliação a ser utilizado.

ConsoleEvaluateClassifier Herda de `EvaluateClassifier` e imprime no console do sistema a saída do processo de avaliação.

CrossValidateEvaluateClassifier Herda de `EvaluateClassifier` e realiza a validação cruzada no *DataSource* especificado pelo parâmetro `evalSource`. Parâmetro obrigatório:
`folds:Integer` - permite definir a quantidade de iterações do processo de validação cruzada.

FileEvaluateClassifier Herda de `EvaluateClassifier` e salva em um arquivo a saída do processo de avaliação. Parâmetro obrigatório:
`file:String` - nome do arquivo de destino, relativo ao diretório `reports` da *KnowledgeBase*.

PosterioriPlaceHolder Esta classe não possui implementação e não interfere no *DataSource*. Ela é utilizada apenas para facilitar a criação de arquivos de configuração que possuem uma hierarquia mais complexa. Através do *place holder* é possível "reservar" um espaço no arquivo de configuração do XML pai, facilitando a inserção de outros elementos no XML filho na posição correta.

RenameSource Herda de `EvaluateClassifier` e renomeia um *DataSource* no `MatcherContext`. Eventualmente é necessário renomear o *DataSource*

PUC-Rio - Certificação Digital Nº 1012636/CA

em um XML de configuração para atender à algum nome específico do XML de configuração definido pelo atributo `parent`.

Parâmetro obrigatório:
`newSource:String` - novo nome do *DataSource* alvo.

SerializeClassifier Herda de `EvaluateClassifier` e persiste um classificador no diretório `classifiers` da *KnowledgeBase* para usos futuros.

Parâmetro obrigatório:
`file:String` - nome do arquivo de destino, relativo ao diretório `classifiers` da *KnowledgeBase*.

PUC-Rio - Certificação Digital Nº 1012636/CA

A.9
Pacote Matcher.Weka.Filter

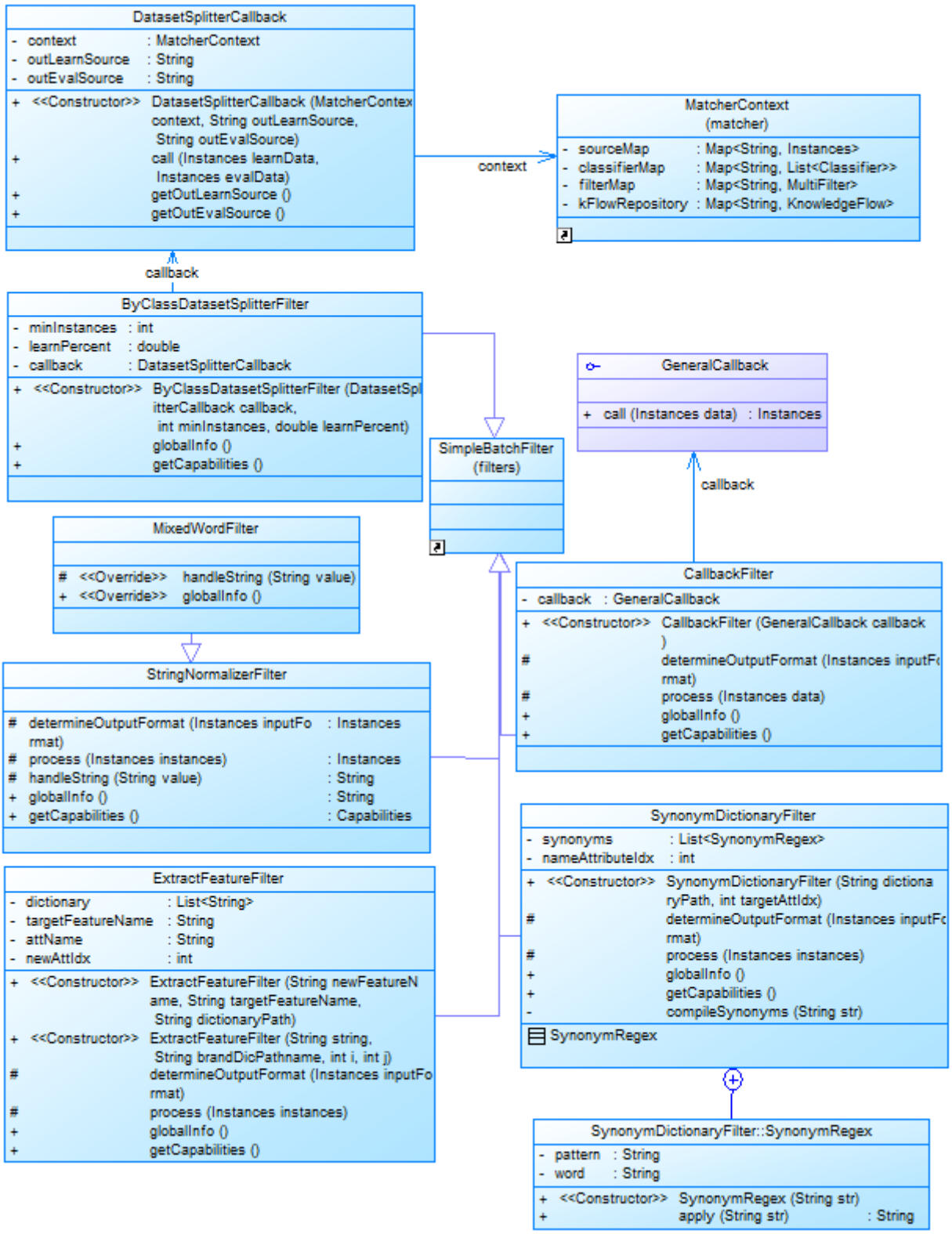

Figura A.9: Diagrama das classes do pacote Matcher.Weka.Filter

PUC-Rio - Certificação Digital Nº 1012636/CA

CallbackFilter O processo de pré-processamento é conduzido pelo *WEKA*, e por isto, temos pouca interferência sobre ele. O `CallbackFilter`, é um filtro que permite cadastrar uma função de retorno (*callback*), onde podemos realizar ações antes ou depois da execução de algum filtro, dando mais controle sobre o processo do *WEKA*. Herda da classe `SimpleBatchFilter` do *WEKA*.

GeneralCallback Interface que define o contrato da função de retorno do `CallbackFilter`.

ByClassDatasetSplitterFilter Herda da classe `SimpleBatchFilter` do *WEKA*, e é onde ocorre o processamento da classe `ByClassDatasetSplitter` (seção A.5).

DatasetSplitterCallback Herda da interface `GeneralCallback`. Possui a implementação da função de Callback da classe `ByClassDatasetSplitterFilter`, chamada ao final do processamento para atualizar o `MatcherContext` com o resultado do filtro.

ExtractFeatureFilter Herda da classe `SimpleBatchFilter` do WEKA, e é onde ocorre o processamento da classe `ExtractFeature` (seção A.5).

StringNormalizerFilter Herda da classe `SimpleBatchFilter` do WEKA. Contém algumas regras básicas para tratamento de *strings*, tais como, padronização de *case* (todas as palavras são colocadas em minúsculo ou maiúsculo) e remoção de acentuação e caracteres especiais.

MixedWordFilter Herda da classe `StringNormalizerFilter`. Separa palavras que contenham letras e números e várias palavras. Exemplo: a palavra `SAMSUNG5500DX` será separada em `{SAMSUNG, 5500, DX}`.

SynonymDictionaryFilter Herda da classe `SimpleBatchFilter` do WEKA, e é onde ocorre o processamento da classe `SynonymReplacer` (seção A.5).

PUC-Rio - Certificação Digital Nº 1012636/CA

A.10
Pacote Matcher.Weka.Evaluation

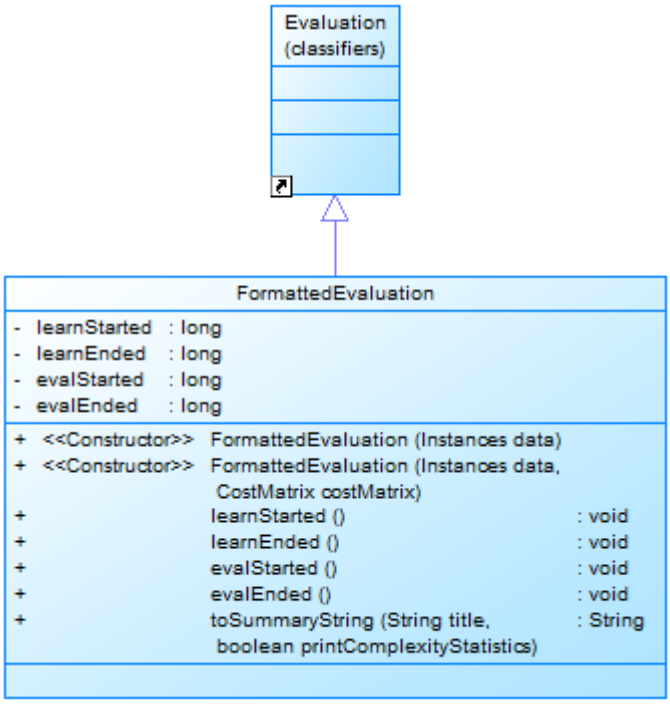

Figura A.10: Diagrama das classes do pacote Matcher.Weka.Evaluation

FormattedEvaluation Permite visualizar as informações do processo de avaliação de forma simplificada e padronizada, exibindo o resultado de cada classificador por linha.

PUC-Rio - Certificação Digital Nº 1012636/CA

A.11
Pacote Debug

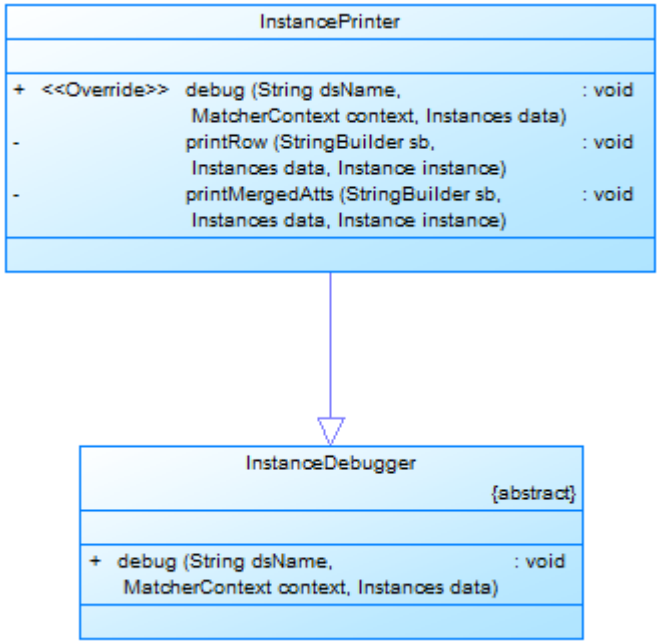

Figura A.11: Diagrama das classes do pacote Matcher.Debug

InstanceDebugger Intercepta cada parte do fluxo de execução, facilitando o processo de *debugging* do *framework*.

InstancePrinter Permite imprimir o estado atual do `MatcherContext` para fins de *debug*.

PUC-Rio - Certificação Digital Nº 1012636/CA

B
XML de Configuração: SPAM

```
<knowledgeFlow name="SpamParent">
 <priori>
   <include name="StringNormalizer" type="com.matcher.kflow.priori.StringNormalizer" enabled="true">
    <configs> <param name="targetSource" value="RawSource"/> </configs>
   </include>
   <include name="ExtractorsPlaceHolder" type="com.matcher.kflow.priori.PlaceHolder" enabled="true"/>
   <include name="StringTokenizer" type="com.matcher.kflow.priori.StringTokenizer" enabled="true">
    <configs>
     <param name="targetSource" value="RawSource"/>
     <param name="targetIndex" value="1"/>
    </configs>
   </include>
   <include name="DatasetSplitter" type="com.matcher.kflow.priori.ByClassDatasetSplitter">
    <configs>
     <param name="targetSource" value="RawSource"/>
     <param name="outLearnSource" value="LearnSource"/>
     <param name="outEvalSource" value="EvalSource"/>
     <param name="learnPercent" value="75"/>
    </configs>
   </include>
 </priori>
 <classifiers>
  <include name="J48" type="com.matcher.kflow.classifier.ClassifierInstantiator" enabled="true">
   <configs>
     <param name="targetSource" value="LearnSource"/>
     <param name="type" value="weka.classifiers.trees.J48"/>
   </configs>
  </include>
  <include name="SMO" type="com.matcher.kflow.classifier.SMOClassifier" enabled="true">
   <configs> <param name="targetSource" value="LearnSource"/> </configs>
  </include>
 </classifiers>
 <posteriori>
  <include name="CrossValidateEvaluateClassifier"
    type="com.matcher.kflow.posteriori.CrossValidateEvaluateClassifier">
   <configs>
     <param name="evalSource" value="RawSource"/>
     <param name="folds" value="2"/>
   </configs>
  </include>
  <include name="EvaluateClassifier" type="com.matcher.kflow.posteriori.ConsoleEvaluateClassifier">
   <configs> <param name="evalDS" value="EvalSource"/> </configs>
  </include>
 </posteriori>
</knowledgeFlow>
```

Listing 1: SpamParent: XML de configuração raiz

PUC-Rio - Certificação Digital Nº 1012636/CA

```
<knowledgeFlow name="SpamEval" extends="SpamParent">
        <exclusions />
        <sources>
                <include name="RawSource"
                        type="com.matcher.kflow.source.ArffInstanceSource" enabled="true">
                  <configs>
                        <param name="arff" value="AllEmails"/>
                        <param name="classIndex" value="0"/>
                  </configs>
                </include>
        </sources>
        <priori>
                <include before="ExtractorsPlaceHolder"
                        name="Synonym" type="com.matcher.kflow.priori.SynonymReplacer">
                  <configs>
                        <param name="targetSource" value="RawSource"/>
                        <param name="targetIndex" value="1"/>
                        <param name="dictionary" value="Synonym"/>
                  </configs>
                </include>
                <include before="ExtractorsPlaceHolder" name="MixedWordModel"
                        type="com.matcher.kflow.priori.FilterInstantiator">
                  <configs>
                          <param name="type" value="com.matcher.weka.filter.MixedWordFilter"/>
                        <param name="targetSource" value="RawSource"/>
                  </configs>
                </include>
        </priori>
</knowledgeFlow>
```

Listing 2: SpamEval: XML de configuração do processo de avaliação

```
<knowledgeFlow name="SpamLearn" extends="SpamParent">
        <exclusions>
                <exclude name="DatasetSplitter" />
                <exclude name="EvaluateClassifier" />
                <exclude name="CrossValidateEvaluateClassifier" />
        </exclusions>
        <posteriori>
                <include before="BeginningPosteriori"
                        name="RenameSource" type="com.matcher.kflow.posteriori.RenameSource" enabled="true">
                        <configs>
                                <param name="targetSource" value="RawSource"/>
                                <param name="newSource" value="LearnSource"/>
                        </configs>
                </include>
                <include name="SerializeClassifier"
                        type="com.matcher.kflow.posteriori.SerializeClassifier" enabled="true">
                        <configs>
                                <param name="fileName" value="SpamModel"/>
                        </configs>
                </include>
        </posteriori>
</knowledgeFlow>
```

Listing 3: SpamLearn: XML de configuração do processo de aprendizado

PUC-Rio - Certificação Digital Nº 1012636/CA

```
<knowledgeFlow name="SpamClassify">
        <merger name="FilterMerger" type="com.matcher.kflow.merger.SerializedFilterMerger">
                <configs>
                        <param name="fileName" value="SpamModel.smo"/>
                </configs>
        </merger>
        <sources>
                <include name="RawSource"
                        type="com.matcher.kflow.source.ArffInstanceSource" enabled="true">
                        <configs>
                                <param name="arff" value="EmailReal"/>
                                <param name="classIndex" value="0"/>
                        </configs>
                </include>
        </sources>
        <classifiers>
                <include name="SMO"
                type="com.matcher.kflow.classifier.ClassifyInstanceClassifier">
                        <configs>
                                <param name="targetSource" value="RawSource"/>
                                <param name="fileName" value="SpamModel.smo"/>
                        </configs>
                </include>
        </classifiers>
</knowledgeFlow>
```

Listing 4: SpamClassify: XML de configuração do processo de classificação

PUC-Rio - Certificação Digital Nº 1012636/CA