Relatório do Trabalho Final de Programação Orientada à Objetos 2

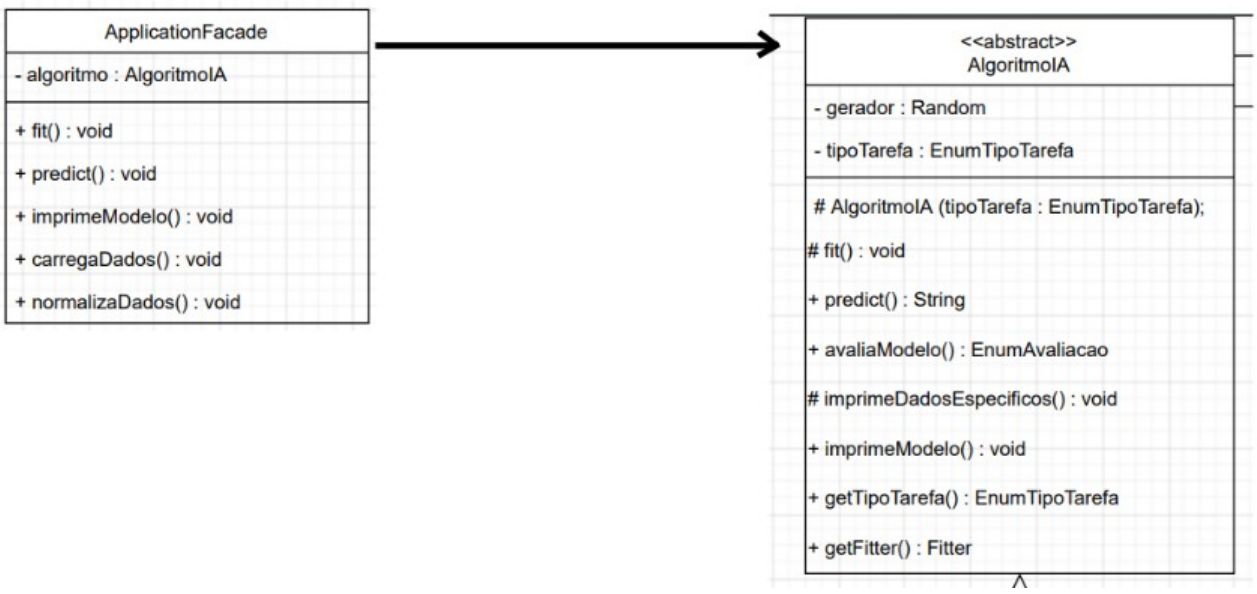
Membro 1: Guilherme Rafael
Membro 2: Paulo Giovany Alves de Oliveira
Membro 3: Otávio Malta Borges

Objetivo do trabalho

Este trabalho tem como objetivo a aplicação de conceitos de padrões de projetos através do desenvolvimento de uma aplicação que possibilita, de forma simulada, a utilização de alguns algoritmos de aprendizado de máquina para treinamento de modelos de aprendizado e análise de dados. Este grupo ficou responsável por desenvolver a aplicação utilizando os padrões de projeto *"Facade"*, *"Template Method"* e *"Strategy".*

O padrão *Facade*

Facade é um padrão de projetos que tem como objetivo ocultar a complexidade de um conjunto de classes da aplicação através da disponibilização de uma interface simples para o uso. Ele isola partes internas do sistema, que podem conter uma série complexa de passos que devem ser seguidos para a execução das funcionalidades, da parte externa (do sistema).



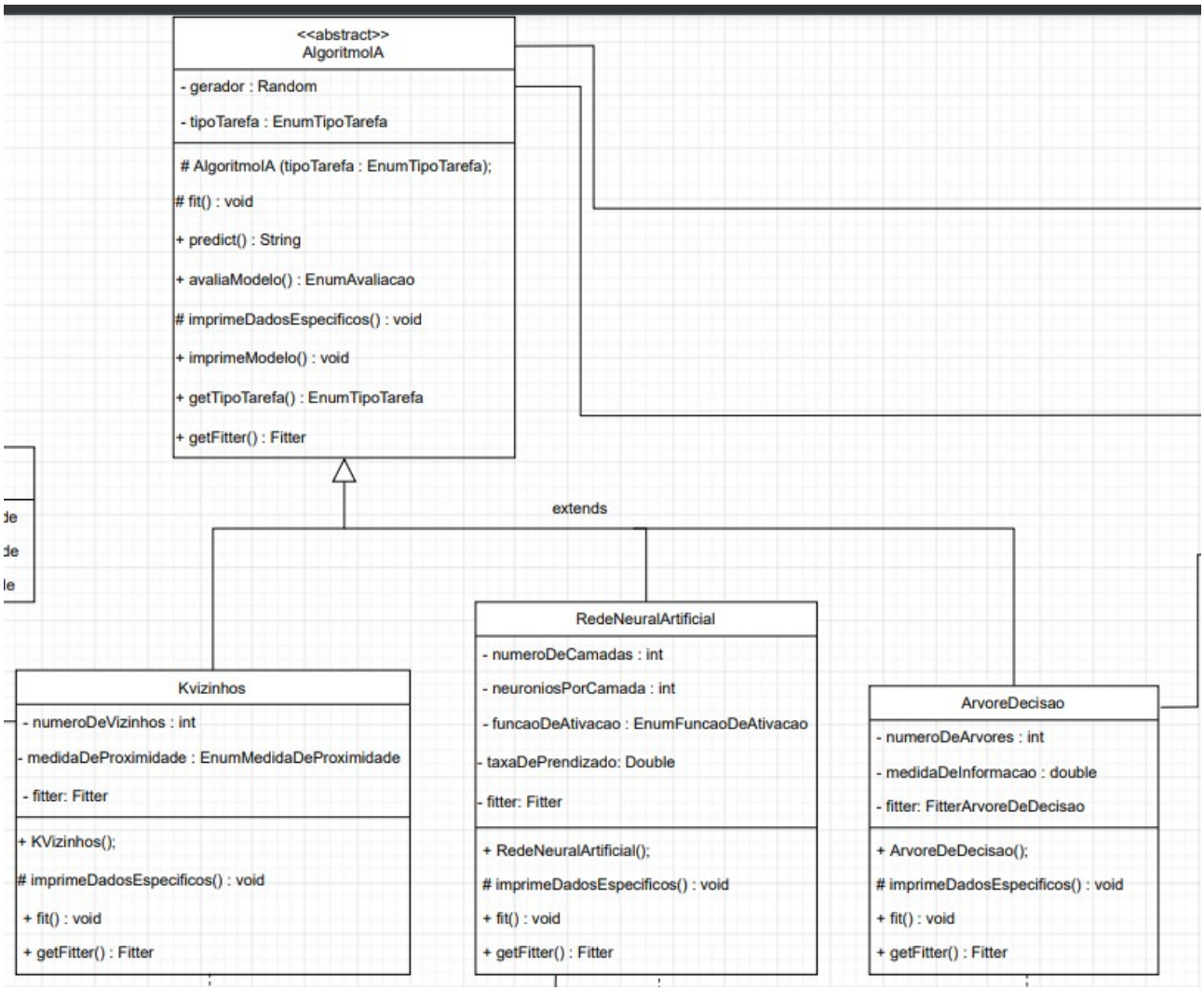
Na implementação da aplicação, o padrão *Facade* foi aplicado através da disponibilização de uma interface (classe *ApplicationFacade*) de acesso aos métodos dos algoritmos de inteligência artificial. A classe expões métodos que dão fácil acesso à execução das principais funcionalidades de todos os algoritmos de IA. Uma vez instanciada de forma correta e configurada para usar determinado algorítimo (ex. K-Vizinhos), a classe permite acessar todos os métodos para o algoritmo selecionado. Para possibilitar isso, ela possui um atributo do tipo AlgoritmoIA, que é uma classe abstrata que

representa, de maneira genérica os algoritmos que podem ser implementados. Para que um novo algoritmo seja implementado na aplicação e poder ser usado na classe *Facade,* ele deve herdar da classe abstrata *AlgoritmoIA*. Assim ela será obrigada a implementar os métodos necessários para funcionar em conjunto com a classe de fachada.

O padrão T*emplate Method*

O padrão *Template Method* tem como objetivo definir um esqueleto de um algoritmo ou processo, de forma que ele executa as partes que são genéricas e delega partes específicas às suas subclasses. Este padrão é bastante aplicado em casos onde o método tem partes similares, porém tem algumas especificidades, que podem ser delegadas às subclasses que executam somente estas.

Na aplicação desenvolvida pelo grupo, esse padrão foi aplicado nas classes *AlgoritmoIA* e suas subclasses *Kvizinhos, RedeNeuralArtificial* e *ArvoreDeDecisao.*



A classe base possui o método *imprimeModelo( ),* que é o método template. Como a classe genérica possui somente algumas informações básicas sobre o algoritmo de IA, ela não consegue executar de forma completa a impressão do modelo. Sendo assim, imprime parte dos dados que ela possui, no caso o tipo da tarefa (Classificação ou Regressão) e obriga que as subclasses implementem o método

imprimeDadosEspecificos( ). Dessa forma, Cada implementação de AlgoritmoIA ter o método que imprime os dados específicos dela.

Método imprimeModelo() na classe AlgoritmoIA

```
protected abstract void imprimeDadosEspecificos();

public void imprimeModelo() {
    System.out.println("Tipo de tarefa: " + this.tipoTarefa.toString());
    System.out.println("Modo: " + this.getFitter().getModo());
    imprimeDadosEspecificos();
}
```

Método imprimeDadosEspecificos() na classe RedeNeuralArtificial

```
@Override
protected void imprimeDadosEspecificos() {
    System.out.println("Número de Camadas: " + numeroDeCamadas);
    System.out.println("Número de Neurônios por Camada: " + neuroniosPorCamada);
    System.out.println("Função de Ativação: " + funcaoDeAtivacao);
    System.out.println("Taxa de Aprendizado: " + taxaDeAprendizado);

}
```

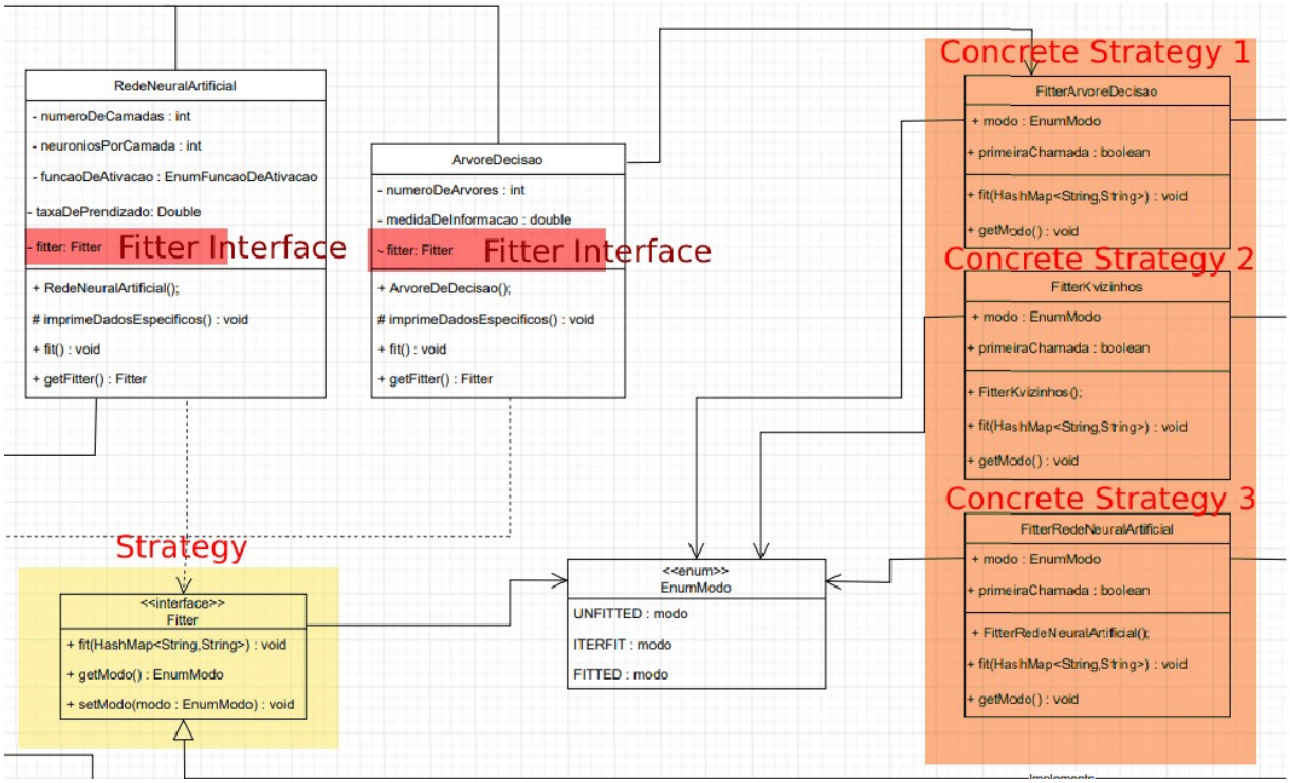
Na imagem acima pode-se observar que na classe base temos o método abstrato *imprimeDadosEspecificos( )* que é implementado na classe *RedeNeuralArtificial* (que herda de AlgoritmoIA) e sobrescreve o método abstrato. Em seguida, no método template *imprimeModelo()* temos a parte genérica do algoritmo, que imprime o tipo de tarefa e o modo de execução, e logo abaixo temos a parte específica, que nesse caso vai ser executada pela subclasse *RedeNeuralArtificial*.

O padrão *Strategy*

O padrão *Strategy* é de propósito comportamental e utilizado para aprimorar a comunicação entre objetos. Neste, define-se uma família de algoritmos (comportamentos), encapsula-se e os torna intercambiáveis e reutilizáveis, permitindo que os algoritmos variem independente dos clientes que os utilizam. Este padrão é geralmente utilizado quando existem classes relacionadas e que se diferem somente no seu comportamento. As estratégias fornecem maneiras de configurar uma classe com um dos comportamentos possíveis.

A utilização desse padrão na nossa aplicação aproveitou-se do método *fit( ),* que é comum à todos os algoritmos de aprendizado de máquina propostos. Pressupondo-se que cada algoritmo tem uma forma específica de fazer o ajuste no seu modelo, foi criada uma interface de nome *Fitter,* que especifica alguns métodos que todo ajustador de modelo de um algoritmo de IA deve possuir. Essa é a nossa Strategy. Em seguida, para que pudessem ser criados métodos concretos que fazem o ajuste do modelo, foram criadas três classes que implementam a interface *Fitter.* Essa classes fazem o ajuste de cada um dos três algoritmos de IA propostos. São elas: *FitterArvoreDecisao, FitterKVizinhos, FitterRedeNeuralArtificial.* Cada uma delas tem implementações diferentes dos métodos descritos na interface *Fitter*. Como próximo passo, nas classes dos algoritmos de IA, foi adicionada uma propriedade genérica de nome fitter e tipo *Fitter* (A interface). Isso possibilita que, caso necessário, um novo ajustador de um novo

algoritmo possa ser para fazer o *‘fit’,* sem a necessidade de alterar a classe do Algoritmo em si, apenas criando uma nova implementação (Concrete Strategy) do Fitter.



Vale citar também que o método *fit( ),* declarado na interface *Fitter*, recebe um *HashMap<String, String>* como parâmetro. Isso permite que sejam passados parâmetros diversos para dentro do método, sem a necessidade de incluí-los na declaração da interface. Dessa forma, basta que os parâmetros sejam passados em formato de um objeto *HashMap* nas chamadas dos métodos concretos de fit( ). Veja o exemplo abaixo:

Declaração do método fit na interface

```
public interface Fitter {

    public void fit(HashMap<String, String> params);
    public EnumModo getModo();
    public void setModo(EnumModo modo);
}
```

Passando parâmetros para o método fit com HashMap

```
@Override
public void fit() {
    HashMap<String, String> params = new HashMap<String, String>();
    params.put("numeroDeArvores",numeroDeArvores.toString());
    params.put("medidaDeInformacao", medidaDeInformacao.toString());
    fitter.fit(params);
}
```

Recuperando os parâmetros dentro da implementação do método

```
@Override
public void fit(HashMap<String, String> params) {
    if(primeiraChamada) {
        System.out.println("Primeira chamada do fit(): passando de UNFITTED para FITTED");
        this.modo = EnumModo.FITTED;
        primeiraChamada = false;
    }

    System.out.println("Ajustando modelo da árvore com as configurações: ");
    System.out.println("-- Número de Camadas: " + params.get("numeroDeCamadas"));
    System.out.println("-- Número de Neuronios por Camadas: " + params.get("neuroniosPorCamada"));
    System.out.println("-- Função de Ativação: " + params.get("funcaoDeAtivacao"));
    System.out.println("-- Taxa de aprendizado: " + params.get("funcaoDeAtivacao"));
}
```

Relação entre padrão de projeto Facade com Template Method:

O padrão de projeto Facade é responsável por disponibilizar uma interface (classe ApplicationFacade) que fornece acesso aos métodos dos algoritmos de inteligência artificial, expondo então os métodos que fornecem fácil acesso à execução das principais funcionalidades de todos os algoritmos de IA. Uma vez instanciada ela permite acessar todos os métodos para o algoritmo selecionado, sendo assim possível utilizar o método Template definido na classe abstrata (imprimeModelo();), que também inclui o espoco da classe “imprimeDadosEspecificos();” dentro dela, ou seja, para cada algoritmo selecionado, disponibiliza saídas diferentes. Logo, ao utilizar a interface de forma correta obtemos também os métodos Template específicos de cada IA que estão definidas dentro das subclasses.

Relação entre padrão de projeto Facade com Strategy:

A utilização do padrão Strategy na nossa aplicação aproveitou-se do método “*fit()”,* que é comum a todos os algoritmos de aprendizado de máquina propostos, porém com cada algoritmo implementado de forma específica o ajuste no modelo. Assim, foi criada uma interface de nome *Fitter,* que especifica alguns métodos que todo ajustador de modelo de um algoritmo de IA deve possuir, essa é a nossa Strategy. Logo, ao

analisarmos a interface Facade “ApplicationFacade” é possível ver que disponibilizamos a utilização do método “fit()”, tal método é especificado em cada ConcreteStrategy ligada à interface Fitter, possibilitando então o isolamento desse método para suas diferentes aplicações e facilitando possibilidade de inserir novos algoritmos e funcionalidades do método “fit()” sem que haja necessidade de especificar qual tipo de “fit()” a aplicação deve seguir. Ademais, temos também a utilização de funções “getModo()” para definir o estado do desempenho (INSATISFATORIO, REGULAR, BOM, MUITO_BOM) e a função “predict()” que também analisa os modos de forma a deduzir as predições.

Relação entre padrão de projeto Strategy com Template Method:

A utilização do método Template “imprimeModelo()” utiliza o método “getModo()”, para imprimir os possiveis modos que um determinado algoritmo se encontra (UNFITTED, ITERFIT, FITTED) que está instanciado na interface Fitter (Strategy). Dessa forma, é possível variar os possíveis comportamento de cada algoritmo seguindo os modos e com o método “getModo()” é possível obter de forma facilitada este dado dentro do método Template “imprimeModelo()”.