Universidade Federal de Uberlândia

Faculdade de Computação

Sistemas de Informação

Trabalho de Programação Orientada a Objetos
Padrões de Projeto em Aprendizado de Máquina

Professor: Murillo G. Carneiro

Alunas:

Beatriz Ribeiro Borges - 12021BSI231

Laura Rosado Rodrigues Muniz - 12021BSI216

Sara Rosado Rodrigues Muniz - 12021BSI215

Uberlândia, 25 de janeiro de 2023

Sumário

Introdução

Nosso projeto visa a utilização de padrões de projeto para o desenvolvimento de soluções fictícias inspiradas em pipelines de aprendizado de máquina e inteligência artificial. Os três padrões estão explicitados a seguir.

Strategy:

O padrão de projeto Strategy é um padrão de propósito comportamental, ou seja, ele é voltado ao algoritmo e a designação de responsabilidades entre objetos. Esse padrão tem como intenção definir uma família de comportamentos, encapsulá-los e torná-los reutilizáveis. Dessa forma, ele permite com que o algoritmo varie independentemente dos clientes que o utilizam.

O Strategy é usado quando temos várias maneiras possíveis de realizar uma tarefa e desejamos poder escolher entre elas de forma fácil e flexível. Ele permite que possamos encapsular cada algoritmo em uma classe separada possibilitando a troca entre eles de forma fácil. Isso é útil quando a lógica de negócios precisa ser alterada dinamicamente ou quando desejamos fornecer aos usuários a capacidade de escolher entre diferentes opções de algoritmos. De modo geral o Strategy é útil quando se tem operações comuns a uma série de sub-classes, mas não é possível o uso de herança de forma eficiente.

Factory Method:

O padrão de projeto Factory Method é um padrão de propósito de criação com escopo de classes, ou seja, ele abstrai e/ou adia o processo de criação dos objetos, mantendo o foco sempre em programar para a interface e não para as implementações. Tal padrão tem como intenção definir uma interface para criar um objeto, mas deixar as subclasses decidirem qual classe instanciar.

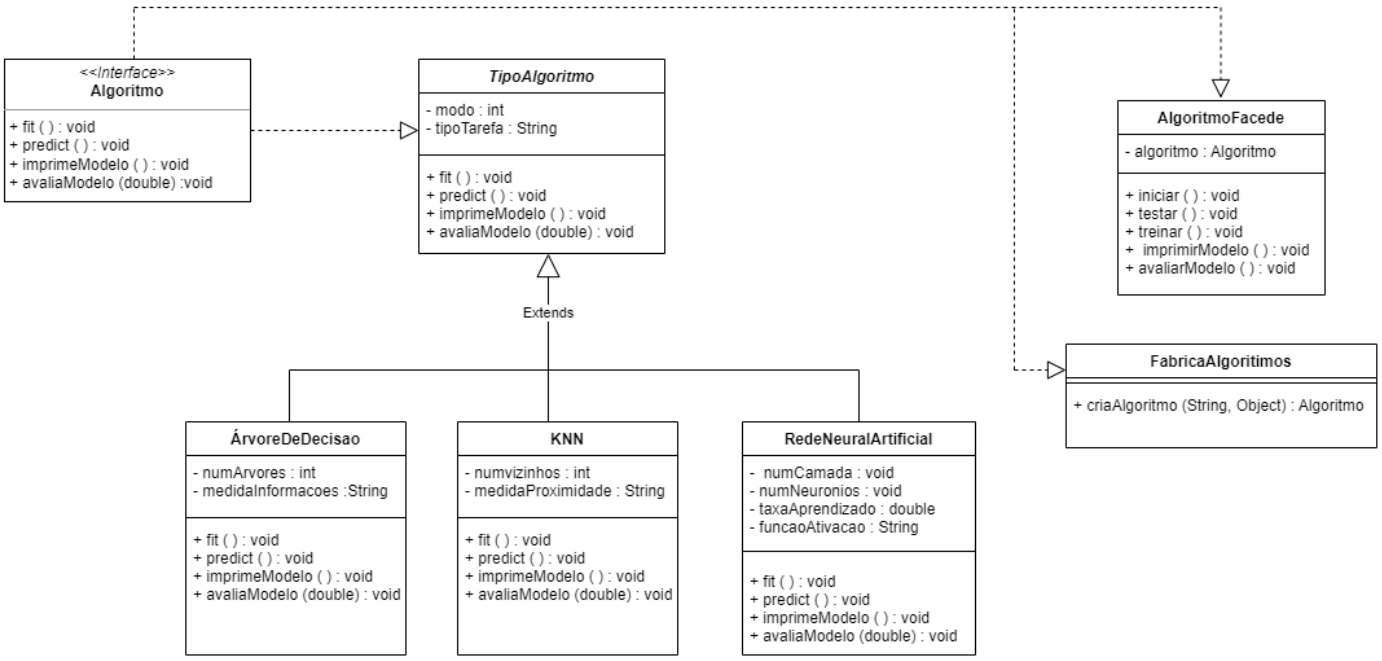
O Método Fábrica é usado quando desejamos criar objetos de uma determinada classe sem especificar a classe concreta que será instanciada. Ele permite que possamos criar objetos a partir de classes abstratas ou interfaces, permitindo que possamos alterar facilmente as classes concretas que são instanciadas sem afetar o código que as utiliza. Isso é útil quando desejamos adicionar novas funcionalidades a um determinado sistema sem precisar modificar o código existente, ou quando queremos criar objetos de forma dinâmica em função de parâmetros de entrada.

Facade:

O padrão de projeto Facade é um padrão de propósito estrutural com escopo de objetos. Ele propõe uma solução para ocultar a complexidade de um conjunto de classes por meio de uma interface simples. Esse padrão de projeto tem como intenção fornecer uma interface unificada para um conjunto de interfaces em um subsistema. Dessa forma, ao definir uma interface de nível mais alto é possível tornar o subsistema mais fácil de usar.

O Facade deve ser usado quando deseja-se fornecer uma interface simples para um subsistema complexo, quando existirem muitas dependências entre clientes e classes de implementação de uma abstração e quando desejamos estruturar seus subsistemas em camadas.

Diagrama de Classes



Desenvolvimento

Este trabalho foi desenvolvido na linguagem JAVA utilizando os três padrões citados: Strategy, Factory e Façade. O trabalho foi pensado para ter a identidade de um padrão de arquitetura MVC. MVC (Model-View-Controller) é um padrão de arquitetura de software que separa a lógica de negócios (Model), a interface do usuário (View) e a lógica de controle (Controller) em camadas distintas. Permitindo, assim, uma maior facilidade de manutenção e extensão do código. Portanto, o projeto foi separado em camadas de forma que o modelo seja independente da visão, assim como do controle. Para tanto foi pensado para o View um “Menu” sendo chamado na Main. No Controller ficam a classe que implementa o menu e a classe “AlgoritmoFacade” (mais a frente será explicada) que é chamada no “Menu”. A implementação mais complexa do código fica destinado ao pacote Modelo.

Começa-se pela interface “Algoritmo” que define quatro métodos: fit(), predict(), imprimeModelo() e avaliaModelo(). As interfaces em Java são como um “contrato” para uma classe, elas especificam um conjunto de métodos que uma classe deve implementar. Será a classe abstrata “TipoAlgoritmo” a fazer a implementação desses métodos citados que depois serão reutilizados nas classes concretas. Com isso, usa-se aqui o padrão Strategy, uma vez que será possível reaproveitar os códigos aplicados nessa interface em diversas classes.

A interface “Algoritmo” também é usada no padrão Factory, sendo ela o Produto. A classe abstrata “TipoAlgoritmo” implementa “Algoritmo”, sendo a representação do Produto Concreto. Nela há uma variável protegida "tipoTarefa" que armazena o tipo de tarefa que o algoritmo irá realizar (regressão, classificação ou rede neural). Foi implementada um tipo de

tarefa extra, rede neural, para que fosse feita uma classificação dos algoritmos pelo tipo da tarefa escolhido. Outra variável protegida "modo" armazena o status do algoritmo: se ele não está treinado recebe 0 (unfitted), treinado 1 (fitted) ou em processo de treinamento 2 (iterfit). A classe também possui os métodos "fit()", "predict()", "imprimeModelo()" e "avaliaModelo()", que são abstratos e serão implementados pelas classes filhas, além de seus getters e setters para as variáveis "tipoTarefa" e "modo".

Ainda no padrão Fábrica, há a classe “FabricaAlgoritmos” que é a implementação do Creator. A classe possui um método estático "criaAlgoritmo" que recebe como parâmetro uma string "tipotarefa" e um número variável de objetos "args". Essa generalização foi feita, pois só será possível saber quais parâmetros deverão ser passados para a instanciação no momento da execução. O método verifica qual é o tipo de tarefa passado e cria uma instância da classe correspondente, passando os argumentos necessários. Se "tipotarefa" for "classificação", cria uma instância de "ArvoreDeDecisao", se for "regressão", cria uma instância de "KNN" e se for "redeneural" cria uma instância de "RedeNeuralArtificial". Se o tipo de tarefa não for reconhecido, o método retorna null.

```java
package Modelo;

public class FabricaAlgoritmos {
    1 usage
    public static Algoritmo criaAlgoritmo(String tipotarefa, Object...args) {
        if (tipotarefa.equalsIgnoreCase( anotherString: "classificacao")) {
            return new ArvoreDeDecisao(tipotarefa, (int) args[0], (String) args[1]);

        } else if (tipotarefa.equalsIgnoreCase( anotherString: "regressao")) {
            return new KNN(tipotarefa,(int) args[0], (String) args[1] );

        } else if (tipotarefa.equalsIgnoreCase( anotherString: "redeneural")) {
            return new RedeNeuralArtificial(tipotarefa, (int) args[0], (int) args[1], (String) args[2], (double) args[3]);
        }
        return null;
    }
}
```

Para as classes concretas, sendo Concrete Creator em Fábrica, foi implementado os métodos fit, predict, imprimeModelo e avaliaModelo. O método fit treina o modelo, o predict realiza a previsão, o imprimeModelo imprime as informações do modelo e o avaliaModelo avalia o desempenho do modelo. Todos esses métodos foram desenvolvidos nos três algoritmos de aprendizado: KNN(K-Vizinhos Mais Próximos), Árvore de Decisão e Rede Neural Artificial, a mudança é na passagem dos atributos. O método Fit, de treinamento, passa uma String avisando do treinamento do algoritmo escolhido se já não estiver no modo fitted. O predict passa uma string avisando do teste no modelo de aprendizagem, caso esse modelo já tenha sido treinado. O método “AvaliaModelo” faz uma classificação a partir de um número randômico indicando se a predição do modelo atual está insatisfatória, regular ou boa/ótima.

```
1 usage
public void fit() {
    if(getModo()==1){
        System.out.println("Dados já ajustados.");
    } else{
        System.out.println("Treinando modelo Arvore de Decisão...");
        setModo(1);
    }


}
1 usage
public void predict() {
    if(getModo()==0){
        System.out.println("Não é possível predizer, faça o treinamento.");
    } else{
        System.out.println("Testando modelo Arvore de Decisão...");
    }
```

```
public void avaliaModelo() {
    double dados = Math.random(); //0.0-1.0
    if(getModo()==0){
        System.out.println("É preciso treinar o modelo");

    } else{
        if(dados<0.3){ //insatisfatório
            setModo(0); //unfitted
            System.out.println("Desempenho Insatisfatório: "+dados);

        } else if (dados<0.7 && dados<1) {  //regular
            setModo(2); //iterfit
            System.out.println("Desempenho Regular: "+dados);

        }else { //bom/ótimo
            setModo(1); //fitted
            System.out.println("Desempenho Bom/Ótimo: "+dados);
        }
    }
}
```

Já no Controller, continuando a estrutura MVC, foi implementado o padrão Façade para haver uma interface “limpa” para o usuário programador ao mesmo tempo que é ele que faz a conexão entre o View e o Modelo. É na classe “AlgoritmoFacade” que são feitas as chamadas principais para o funcionamento do programa. Ela é responsável por criar uma instância específica do algoritmo de acordo com o tipo de tarefa e os parâmetros passados. Para que ela consiga fazer a ligação entre os dois modos (View e Modelo) é usado um atributo do tipo Algoritmo.

```java
package Controller;
import Modelo.*;

2 usages
public class AlgoritmoFacade {
    5 usages
    private Algoritmo algoritmo;

    3 usages
    public void iniciar(String tipoTarefa, Object...args) {
        algoritmo = FabricaAlgoritmos.criaAlgoritmo(tipoTarefa, args);
    }
    1 usage
    public void treinar() {
        algoritmo.fit();
    }
    1 usage
    public void testar() {
        algoritmo.predict();
    }
    1 usage
    public void imprimirModelo() {
        algoritmo.imprimeModelo();
    }
    1 usage
    public void avaliarModelo() { algoritmo.avaliaModelo(); }
```

No Controller foi implementada o Menu que chama as funções da classe “AlgoritmoFaçade”. Por fim, no pacote View está a Main contendo a chamada da função Menu. Foi feita a escolha de deixar tanto o “AlgoritmoFaçade”, quanto a implementação do “Menu” no Controller, pois há uma maior organização dos códigos, uma vez que há muitas chamadas de funções em cada uma dessas classes. Quando pensa-se no usuário programador, ele entende o

Façade, mas para o usuário comum final é interessante deixar apenas o menu como opção de execução, escondendo as chamadas de funções e instanciações.

Referências

Slides da matéria disponíveis no Microsoft Teams.