INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO SUDESTE DE MINAS GERAIS
CAMPUS RIO POMBA

ESPECIALIZAÇÃO EM DESENVOLVIMENTO WEB E MOBILE

Anderson José Simplício

API de Diagnóstico de Diabetes com Aprendizado de Máquina: Precisão e Confiabilidade em Saúde

Anderson José Simplício

RIO POMBA
2023

API de Diagnóstico de Diabetes com Aprendizado de Máquina: Precisão e Confiabilidade em Saúde

Trabalho de conclusão de curso apresentado ao Departamento do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais, *campus* Rio Pomba, como parte dos requisitos para a obtenção do título de Especialista em Desenvolvimento Web e Mobile.

Orientador (a): LUCAS GRASSANO LATTARI

Rio Pomba

2023

Verificar disposição da Seção de Catalogação e Classificação da Biblioteca do *campus* do IF onde o trabalho foi desenvolvido (ou setor equivalente)

Ficha catalográfica

(Informação a ser adicionada no verso da Folha de Rosto)

Adicionar a ficha catalográfica confeccionada na Seção de Catalogação e Classificação da Biblioteca do *campus* (ou setor equivalente) do IF onde o trabalho foi desenvolvido.

Anderson José Simplício

API de Diagnóstico de Diabetes com Aprendizado de Máquina: Precisão e Confiabilidade em Saúde

Trabalho de Conclusão de Curso apresentado ao Departamento do Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais, *Campus* Rio Pomba, como parte dos requisitos para a obtenção do título de Especialista em Desenvolvimento WEB/Mobile.

Aprovado em: ____ / ____ / _____.

BANCA EXAMINADORA

_____

Prof.(a) xxxxxxxxxxxxx
Especialista/Mestre/Doutor(a) em
Instituição

_____

Prof.(a) xxxxxxxxxxxxxxxxxxxx
Especialista/Mestre/Doutor(a) em
Instituição

_____

Prof.(a) xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Especialista/Mestre/Doutor(a) em
Instituição

Dedico este trabalho a minha mãe, que sempre me incentivou a seguir em frente.

AGRADECIMENTOS

A Deus, inteligência Suprema do Universo e causa primária de todas as coisas.

Dedico este trabalho à minha amada esposa, Edna Mendes, e ao nosso precioso filho, Arthur Mendes e Simplício.

Em especial reconhecimento ao meu orientador, Lucas Lattari, cuja dedicação incansável desempenharam um papel crucial na concretização deste projeto.

Dedico este tributo de gratidão ao Professor Félix Eustáquio Basílio, cuja fé em minha capacidade e generosidade ao abrir as portas do cursinho pré-vestibular foram luzes que iluminaram meu caminho educacional.

Aos professores e aos colegas do Instituto Federal de Educação Ciência e Tecnologia do Sudeste de Minas – Campus Rio Pomba.

“A ciência não é somente compatível com a espiritualidade; ela é uma fonte profunda de espiritualidade”. (Carl Sagan)

RESUMO

A análise de dados na saúde, especialmente no diagnóstico de doenças como o diabetes, tem evoluído com o uso de tecnologias de aprendizado de máquina. Este avanço abre caminho para a aplicação prática dessas tecnologias, respondendo à crescente demanda por soluções de IA facilmente integráveis em ambientes *web*. Com esse objetivo, o presente estudo concentra-se na implementação de modelos de aprendizado de máquina em *frameworks Web*, como Django, para o desenvolvimento de uma API eficiente e acessível para o diagnóstico de diabetes, facilitando a adoção por desenvolvedores e a utilização prática pela população. Para tal, exploramos métodos como Floresta Randômica, Árvore de Decisão e Máquinas de Vetores de Suporte, com a otimização dos modelos realizada via *GridSearchCV*. A integração desses modelos em um ambiente de produção destaca a importância de uma abordagem interdisciplinar para superar a lacuna entre teoria e prática. Os resultados demonstram a eficácia da Floresta Randômica no diagnóstico de diabetes, com a API testada para garantir a precisão das previsões e a funcionalidade adequada. Este trabalho não apenas evidencia o potencial dos modelos de aprendizado de máquina na saúde, mas também discute os desafios e soluções práticas para sua aplicação.

Palavras-chave: Diabetes, Aprendizado de Máquina, Floresta Randômica, Árvore de Decisão, Máquinas de Vetores de Suporte, Pré-processamento de Dados, API, Django.

ABSTRACT

Diabetes Diagnosis API with Machine Learning: Precision and Reliability in Healthcare

Data analysis in healthcare, especially in diagnosing diseases such as diabetes, has significantly advanced with the use of machine learning technologies. This progress paves the way for practical applications of these technologies, meeting the growing demand for machine learning solutions easily integrated into web environments. In this context, the current study focuses on implementing machine learning models in web frameworks, such as Django, to develop an efficient and accessible API for diabetes diagnosis. This facilitates adoption by developers and practical use by the population. Key methods, including Random Forest, Decision Tree, and Support Vector Machines, are explored, with optimization through GridSearchCV. This highlights the importance of an interdisciplinary approach for the effective integration of these models in production environments. The results underscore the effectiveness of Random Forest in diagnosing diabetes and confirm the accuracy and functionality of the API. This work not only demonstrates the potential of machine learning models in healthcare but also addresses the challenges and practical solutions for their everyday application.

Keywords: Diabetes, Machine Learning, Random Forest, Decision Tree, Support Vector Machines, Data Pre-processing, API, Django.

LISTA DE ILUSTRAÇÕES

LISTA DE ILUSTRAÇÕES

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O avanço da tecnologia e o volume crescente de dados disponíveis têm feito da análise de dados um campo cada vez mais relevante em diversas áreas do conhecimento, incluindo a saúde. Esse progresso tem sido especialmente significativo para o diagnóstico e tratamento de diversas enfermidades. Um editorial da revista *Nature*, publicado em 2020, destacou como as novas tecnologias estão impulsionando inovações em pesquisa clínica, oferecendo novas soluções para ensaios clínicos e reunindo uma diversidade de populações para estudos (*Nature*, 2020, p. 1).

Isso demonstra que a coleta e análise de dados médicos têm se mostrado promissoras para aprimorar a precisão do diagnóstico de doenças. O diagnóstico de diabetes é um exemplo concreto do impacto positivo que a análise de dados pode ter na medicina.

Tradicionalmente, o diagnóstico dessa doença dependia principalmente de exames de laboratório, como testes de glicose no sangue. No entanto, esses métodos muitas vezes não identificavam casos iniciais ou subclínicos da doença.

De acordo com as Diretrizes da Sociedade Brasileira de Diabetes (2020, p. 20), a doença frequentemente se manifesta de forma assintomática ou com sintomas leves por extensos períodos. Geralmente, o diagnóstico é feito através de exames laboratoriais de rotina ou quando já existem complicações crônicas manifestas.

> Com o apoio do aprendizado de máquina, o diagnóstico precoce tornou-se possível. Indivíduos de alto risco podem agora tomar medidas de precaução para evitar as consequências da doença pelo maior tempo possível. O diagnóstico precoce bem-sucedido depende em grande parte da seleção precisa de classificadores e características relacionadas (Aftab *et al*., 2021).

Cubillos *et al*. (2023) apresentaram modelos de aprendizado de máquina para antecipar casos de diabetes durante a fase inicial da gravidez. Os modelos provaram ser eficazes na detecção precoce da condição, contribuindo para um cuidado mais apropriado às gestantes.

No entanto, apesar da perspectiva de aprimorar a acurácia do diagnóstico de enfermidades, a incorporação de modelos de aprendizado de máquina em ambientes de produção se revela uma empreitada complexa. Isso ocorre porque existe uma significativa disparidade entre a teoria de *machine learning* e sua concretização em produtos reais. Poucas organizações dispõem de uma metodologia claramente definida para o

desenvolvimento e operacionalização de produtos que empregam essa tecnologia, especialmente nos casos em que aperfeiçoamentos contínuos se fazem necessários.

> Um projeto de *machine learning* em produção também necessita de um acompanhamento especial. Em um desenvolvimento de *software* tradicional, garantir que seu produto esteja disponível e funcional para os clientes que utilizam pode ser suficiente. Já em um projeto de *machine learning* é necessário que seu modelo continue dentro das suas métricas definidas na construção do negócio, portanto esse acompanhamento é vital para o projeto (Mello *et al*., 2021, p. 11).

Adicionalmente, a área da ciência de dados é notória por sua ênfase primordial na criação de algoritmos de aprendizado de máquina e na construção de modelos, utilizando ferramentas de mineração de dados, acompanhado de linguagens de programação especializadas como *Python* e R.

> Não existe uma fórmula ou procedimento claro para a integração de modelos de aprendizado de máquina (ML) com aplicações *Web* criadas com *Django* ou *Flask.* Isso implica que, embora a maioria dos cientistas de dados seja habilidosa na criação de modelos de ML usando ferramentas de mineração de dados, muito poucos são proficientes na criação dos mesmos modelos usando linguagens como R ou Python (Hadullo e Getuno, 2021).

Conforme Hadullo e Getuno (2021), o desafio se intensifica devido à necessidade de integrar uma aplicação *Web* com o aprendizado de máquina, formando uma única solução.

Embora as tecnologias de aprendizado de máquina ofereçam grande potencial para melhorar o diagnóstico de doenças, como o diabetes, há desafios significativos na implementação desses modelos em ambientes de produção. Além disso, há uma lacuna na integração eficiente desses modelos com aplicações *Web* existentes. Dessa forma, o foco desta monografia é o desenvolvimento e integração de modelos de aprendizado de máquina em aplicações *Web*, com ênfase particular no diagnóstico de diabetes.

2 OBJETIVOS

O objetivo principal deste trabalho é desenvolver e avaliar modelos de aprendizado de máquina para a detecção de diabetes, visando identificar o modelo que apresenta as melhores métricas, com base nos conjuntos de dados selecionados para este estudo.

Posteriormente, o estudo abordará o desenvolvimento de uma arquitetura de *software* e a construção de um *pipeline* de implantação, empregando o *Django Rest Framework* (DRF) em *Python*, para integrar os modelos de aprendizado de máquina desenvolvidos.

2.1 OBJETIVOS ESPECÍFICOS

- Coletar e processar conjuntos de dados relacionados ao diabetes para treinar e validar os modelos de aprendizado de máquina.
- Desenvolver e aprimorar algoritmos de aprendizado de máquina que sejam eficazes na identificação de indicadores biomédicos relevantes para a detecção de diabetes.
- Comparar a eficácia de diversos algoritmos em relação à acurácia, sensibilidade e especificidade na detecção de diabetes.
- Elaborar uma arquitetura de *software* otimizada para a integração eficiente dos modelos de aprendizado de máquina em sistemas de saúde digitais já existentes.
- Desenvolver um fluxo de trabalho de implantação para tornar o modelo acessível e utilizável em ambiente de produção.
- Implementar a integração do modelo de aprendizado de máquina mais eficaz com o *Django Rest Framework*, para facilitar o desenvolvimento de uma aplicação *Web* robusta ou de uma API específica.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo fornecer os fundamentos teóricos necessários para a compreensão do trabalho. Serão abordadas diversas tecnologias e conceitos essenciais em áreas como interfaces de programação de aplicativos (APIs), arquitetura *REST*, *frameworks* de desenvolvimento *Web* como *Django* e conceitos de aprendizado de máquina e ciência de dados. Cada subseção detalhará um desses tópicos para esclarecer sua relevância e aplicação no contexto deste estudo.

3.1 FERRAMENTAS COMPUTACIONAIS

Essa subseção aborda as aplicações de software usadas no desenvolvimento deste trabalho.

3.1.1 API (*Application programming interface*)

Sobre o conceito de API, podemos afirmar que:

> API é a sigla em inglês para *Application Programming Interface*, ou interface de programação de aplicações. As interfaces de programação de aplicativos (APIs) são conjuntos de ferramentas, definições e protocolos para a criação de aplicações de software. APIs conectam soluções e serviços, sem a necessidade de saber como esses elementos foram implementados (Red Hat, 2021).

Assim, uma API age como um conector que padroniza a comunicação entre sistemas. Sob essa ótica, APIs podem ser vistas como um contrato de serviço entre duas aplicações, estabelecendo as regras para a comunicação. Este contrato é feito através de uma série de solicitações e respostas que são guiadas por uma documentação específica.

Sobre a documentação de software:

> A documentação de um software é composta por várias partes diferentes que abrangem todo o sistema e pode ser dividida em dois grandes grupos: documentação técnica e documentação de uso. A primeira é voltada ao desenvolvedor ou pessoa de TI e compreende principalmente dicionários e modelos de dados, fluxogramas de processos e regras de negócios, dicionários de funções e comentários de código.Já a documentação de uso é voltada tanto para o usuário final quanto para o administrador do sistema e comumente é formada por apostilas ou manuais que apresentam como o software deve ser usado, o que esperar dele e como receber as informações que se deseja (Michelazzo, 2006).

Em uma arquitetura cliente e servidor, APIs recebem parâmetros e retornam resultados. Para ilustrar, considere o serviço de correios: ele possui uma API que aceita um CEP como entrada e retorna um endereço como saída. Este modelo assegura que haja uma distribuição equilibrada das responsabilidades entre o cliente e o servidor, garantindo independência e escalabilidade.

As APIs geralmente operam usando métodos HTTP para interagir com recursos na Web. Os verbos mais comuns são: *GET, POST, PUT* e *DELETE*. Respectivamente, esses verbos são usados para recuperar dados, criar dados, atualizar informações existentes e remover informações.

Essas operações permitem que sistemas distintos e em diferentes plataformas possam se comunicar de forma eficiente. A comunicação via API, realizada por meio desses verbos HTTP, geralmente ocorre em um contexto sem estado, não mantendo uma sessão persistente entre as interações (Beckenkamp, 2016).

3.1.2 *REST*

O *Representational State Transfer* (REST) representa uma abordagem arquitetônica para o design e desenvolvimento de serviços *web*, focando na eficiência da comunicação entre sistemas distintos. Esta permite que sistemas diferentes na *Internet* conversem entre si de forma eficiente. Assim, REST pode ser entendida como um conjunto de melhores práticas para a construção de API.

Para compreender o conceito, é importante destacar a noção de *hipermídia*. *Hipermídia* refere-se à integração de diferentes formatos de mídia, como texto, imagens e *multimídia*, em uma estrutura interconectada, exemplificada por páginas *web* com *links* clicáveis. *Hipertexto*, sendo um subconjunto da *hipermídia*, foca nos

aspectos textuais e nos *links* que interligam conteúdos e páginas *web*(Goularte *et al.*, 2000).

Segundo Fielding (2000), o *REST* não está preocupado com os detalhes técnicos da implementação dos componentes nem com a sintaxe dos protocolos. Em vez disso, o foco está na maneira como os diferentes componentes interagem e interpretam os dados.

De acordo com Beckenkamp (2016), o *REST* busca tornar diferentes sistemas compatíveis na Internet. Para ser mais específico, serviços da web que seguem os princípios *REST* são chamados de *RESTful*. Esses serviços empregam um padrão de operações HTTP (como GET, POST, PUT e DELETE) em um contexto sem estado, onde cada requisição é independente das outras, para permitir essa comunicação. Por exemplo, pode-se supor um serviço que fornece informações meteorológicas. Um serviço *RESTful* poderia permitir que um aplicativo móvel solicitasse a previsão do tempo para uma localidade específica e recebesse esses dados em um formato facilmente interpretável.

Em resumo, a API *REST* simplifica a forma como sistemas se comunicam na *Internet*, fazendo uso eficiente da hipermídia e seguindo princípios arquiteturais bem definidos. Isso permite aos desenvolvedores separar as interfaces do usuário do acesso aos recursos, tornando os sistemas mais modulares e fáceis de gerenciar.

3.1.3 *Django*

Segundo o site da Mozilla Developer Network (2023), *Django* é um *framework* de desenvolvimento *Web* de alto nível, escrito na linguagem de programação Python, que permite a criação de sites seguros e de fácil manutenção. Sua abordagem opinativa oferece um conjunto de componentes pré-definidos para lidar com a maioria das tarefas de desenvolvimento *Web.*

No entanto, ainda que adote uma abordagem opinativa, sua arquitetura flexível permite a personalização e extensão, incluindo a adição de novos componentes ou suporte a tecnologias diferentes, conforme necessário.

Segundo Augusto (2021), um *framework* opinativo estabelece diretrizes para a construção de componentes e serviços, incluindo a criação de rotas e injeção de dependência[1]. Neste último caso, um componente ou objeto recebe suas dependências externas, em vez de criá-las internamente.

Django utiliza a arquitetura MTV (*Model-View-Template*), uma adaptação do padrão MVC (*Model-View-Controller*), promovendo efetiva separação de responsabilidades nas aplicações.

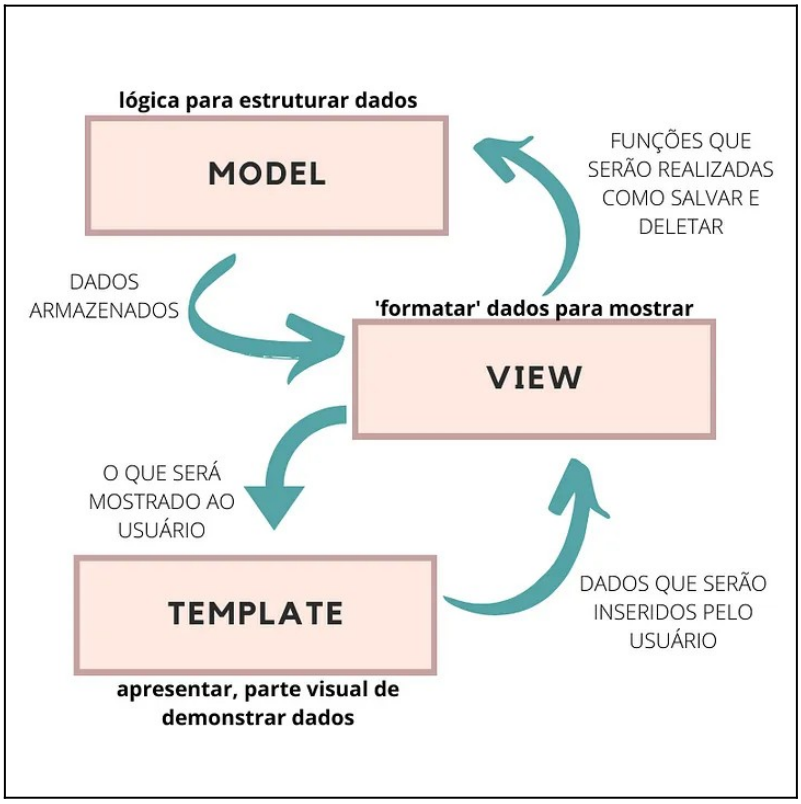
Silva (2022) observa que, apesar das semelhanças com o MVC, as principais diferenças do MTV no *Django* estão na nomenclatura e na interconexão das camadas.

Na camada *Model*, o *Django* gerencia a lógica de negócios e de dados, lidando com a manipulação, extração e persistência das informações. A camada *View* define as rotas e controladores, processando as requisições dos usuários e interagindo com o *Model* para recuperar ou modificar dados conforme o necessário.

Por fim, a camada *Template* foca na apresentação dos dados, empregando arquivos *HTML* enriquecidos com a linguagem de *template* do *Django*, que exibe os dados processados pela *View*. (Figura 1).

[1] A injeção de dependência é um padrão de projeto de *software* que permite que um objeto receba suas dependências de outros objetos, em vez de criá-las ou procurá-las por si mesmo. Isso torna o código mais flexível e reutilizável, pois facilita a substituição e o gerenciamento dessas dependências sem modificar o próprio objeto. Fonte: https://www.dio.me/articles/injecao-de-dependencia

Figura 1 – Arquitetura MTV do *Django*: A camada de modelo gerencia a estrutura de dados e lógica de negócios. A camada de visualização processa requisições e interage com o modelo para obter e formatar dados. Finalmente, a camada de *Template* exibe os dados ao usuário, com foco na representação visual.



Fonte: Silva (2022).

3.2 APRENDIZADO DE MÁQUINA

O aprendizado de máquina, um subcampo da inteligência artificial e da ciência da computação, envolve o desenvolvimento de modelos computacionais que analisam dados para identificar padrões e realizar previsões ou decisões. Esses modelos se adaptam automaticamente com a introdução de novos dados e são aplicados em diversas áreas, desde saúde até *marketing*.

De acordo com Russell e Norvig (2022), o aprendizado de máquina se refere à capacidade de um computador em criar um modelo baseado em dados que pode então ser utilizado como uma hipótese sobre o mundo.

No aprendizado supervisionado, o modelo aprende a associar entradas, como variáveis independentes ou características, as saídas específicas, conhecidas como variáveis dependentes ou rótulos. Isso é feito por meio do uso de dados rotulados, onde cada entrada já está vinculada a um rótulo correspondente. O objetivo principal

é capacitar o modelo a prever corretamente os rótulos para novos conjuntos de dados, baseando-se no conhecimento adquirido durante o treinamento.

Para que seja possível avaliar de forma imparcial o desempenho da previsão, é necessário dividir o conjunto de dados aleatoriamente em três subconjuntos:

- O conjunto de treinamento é usado para ajustar o modelo e encontrar os pesos ou coeficientes ideais para algoritmos como regressão linear, regressão logística, redes neurais artificiais, dentre outros.
- O conjunto de validação é utilizado para avaliar, de forma imparcial, o modelo durante o ajuste de hiperparâmetros. Por exemplo, quando se busca o número ideal de neurônios em uma rede neural artificial ou o melhor *kernel* para um método baseado em máquina de vetores de suporte, experimenta-se valores diferentes. Para cada configuração de hiperparâmetros considerada, ajusta-se o modelo com o conjunto de treinamento e avalia-se seu desempenho com o conjunto de validação.
- O conjunto de teste é necessário para uma avaliação imparcial do modelo final.

O aprendizado de máquina é uma área ampla e composta por uma série de técnicas complexas. Dentre essas, uma estratégia considerada notável é o *ensemble learning*, que melhora a precisão das previsões ao combinar os resultados de vários modelos distintos.

3.2.1 Regressão linear

A regressão linear é frequentemente empregada para estabelecer uma relação entre variáveis independentes (características dos objetos) e uma variável dependente (o rótulo a ser predito), sendo comum em contextos onde a variável dependente é contínua. O algoritmo modela essa relação através de uma função linear, permitindo fazer previsões acuradas sobre novos dados.

Ela também pode ser aplicada para analisar e prever valores associados a parâmetros específicos. No contexto deste estudo, esse método foi usado para computar o colesterol e o HBA1C (marcador tipicamente usado para avaliar a média dos níveis de glicose do sangue em pessoas com diabetes).

3.2.2 *Dummy classifier*

O *Dummy Classifier* é um classificador que sempre prevê a mesma classe, baseado em um conjunto de regras predeterminadas, não relacionadas às características dos dados de entrada. Embora simples, este é uma ferramenta útil para estabelecer uma linha de base na comparação com classificadores mais complexos.

Ao comparar o desempenho do *Dummy Classifier* com os outros classificadores, é possível obter uma ideia da melhoria que esses classificadores podem trazer. Caso o *Dummy Classifier* tenha um desempenho semelhante ou superior a outros métodos, isso significa que as características de entrada não são suficientes para distinguir as classes de forma eficaz.

O *Dummy Classifier* escolhe a classe de acordo com uma estratégia específica. As estratégias tipicamente disponíveis estão descritas na Tabela 1.

Tabela 1 — Resumo das estratégias de classificação utilizadas pelo *Dummy Classifier*, suas descrições e exemplos de aplicação.

Estratégia	Descrição	Exemplo
Constant	Prevê sempre a mesma classe, independentemente dos recursos de entrada.	Se a classe a ser prevista for 1, o classificador sempre preverá 1, mesmo que os recursos de entrada sejam diferentes.
Most_frequent	Prevê a classe mais frequente no conjunto de treinamento.	Se a classe mais frequente no conjunto de treinamento for 1, o classificador preverá 1 para todos os exemplos no conjunto de teste.
Stratified	Faz previsões que refletem as proporções das classes no conjunto de treinamento, mas de forma aleatória respeitando essas proporções.	Se no conjunto de treinamento a classe 1 ocorre em 70% das instâncias e a classe 0 em 30%, então, ele preverá a classe 1 para aproximadamente 70% das amostras e a classe 0 para os restantes 30%, mantendo as proporções observadas no treinamento
Uniform	Prevê as classes de forma aleatória, com igual probabilidade para cada classe.	Nesse caso, o classificador prevê as classes aleatoriamente, sendo 1 com probabilidade de 50% e 0 com probabilidade de 50%, independentemente dos dados de entrada.

Fonte: Elaborado pelo autor.

É crucial destacar que, ao utilizar o *Dummy Classifier* como *benchmark* (*baseline* para comparação entre trabalhos), podemos avaliar se os classificadores avançados oferecem melhorias significativas na precisão da previsão. Esta análise comparativa é fundamental para entender o valor e a eficácia dos modelos de *machine learning* implementados neste estudo.

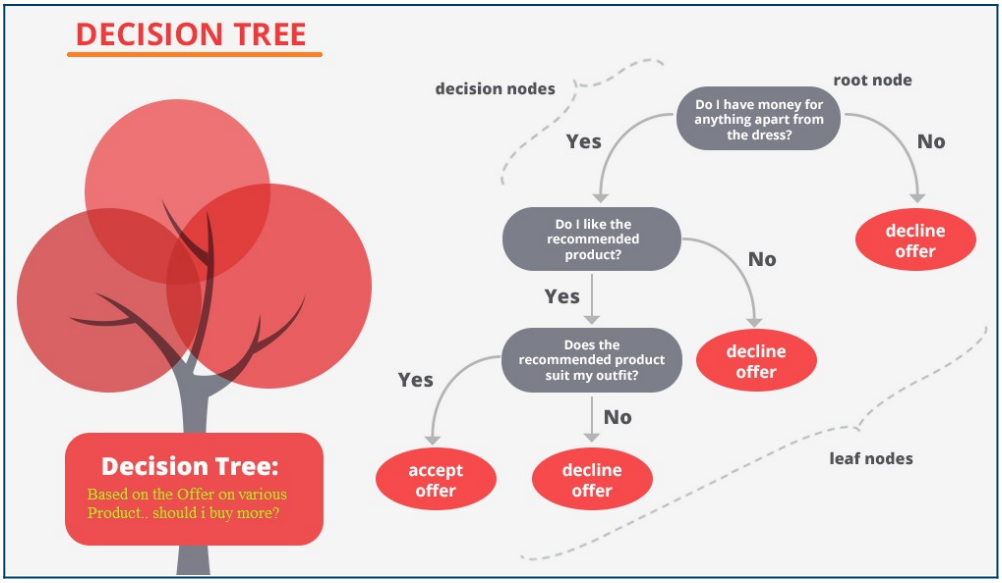
3.2.3 Árvores de decisão e florestas aleatórias

Uma árvore de decisão é um modelo de aprendizado de máquina que representa uma função a partir de um vetor de valores de atributos (entradas), realizando uma série de testes para chegar a uma decisão final (saída). Conforme Russell e Norvig (2022), estas decisões são tomadas com base em testes de atributos dos vetores de entrada, começando em um nó raiz e progredindo até os nós folha mais externos, que representam as decisões finais. As árvores de decisão são populares e versáteis, sendo empregadas em tarefas como classificação, regressão e agrupamento.

A Figura 2 ilustra uma árvore de decisão voltada para a classificação entre duas classes: “Aceitar a oferta” e “Não aceitar a oferta”. Esta árvore é estruturada com nós, que determinam decisões, e folhas que indicam as classes resultantes. Considerando o exemplo abaixo, para o nó raiz, é feita a pergunta: “Tenho dinheiro para alguma coisa além do vestido?”. Uma resposta afirmativa leva ao ramo esquerdo, culminando na próxima escolha, enquanto uma negativa direciona ao ramo direito, resultando em um nó folha que recusa a oferta.

A árvore descrita é classificada como binária, pois cada nó divide os dados em dois subconjuntos específicos, ou seja, duas possibilidades. No entanto, há árvores de decisão multinárias capazes de segmentar os dados em múltiplos grupos.

Figura 2 — Esta figura ilustra uma árvore de decisão relacionada à decisão de comprar um produto com base em diversas considerações. A partir do nó raiz, é questionado se o indivíduo possui dinheiro para o produto. Caso não possua, isso leva diretamente à decisão de “recusar a oferta”. Se a resposta for afirmativa, avalia-se a preferência pelo produto recomendado e, posteriormente, sua compatibilidade com o visual do indivíduo. Os ramos terminam em nós folha que representam a decisão final: “oferta aceita” ou “oferta recusada”. Os nós de decisão intermediários, indicados como nós de decisão, guiam o processo de escolha através de perguntas sequenciais.



Fonte: Sacramento (2023).

Esse é um esquema simples de árvore de decisão. Modelos mais complexos podem possuir maior profundidade e mais nós e folhas, utilizando diferentes critérios para particionar o conjunto de dados.

Após compreender as árvores de decisão, podemos avançar para o método de floresta aleatória. Segundo Breiman e Cutler (2001), a floresta aleatória é um algoritmo de aprendizado de máquina que combina as saídas de várias árvores de decisão para produzir um resultado final mais preciso ou robusto. Este método é considerado um tipo de *ensemble* e melhora a precisão e a robustez em comparação com uma única árvore de decisão.

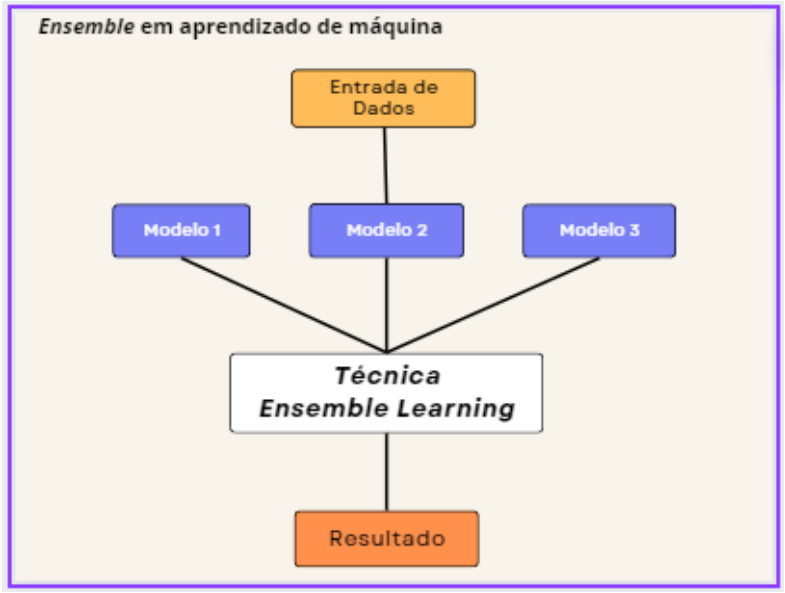
Dentro de uma floresta aleatória, cada árvore de decisão é gerada e treinada usando algoritmos como o CART (*Classification and Regression Tree*), com cada uma operando em um subconjunto aleatório dos dados para aumentar a diversidade e a robustez do modelo final. A facilidade de uso e a flexibilidade da floresta aleatória a tornaram popular, sendo capaz de lidar com problemas de classificação e regressão. As árvores individuais na floresta utilizam critérios de divisão baseados nos atributos para segmentar o espaço dos dados, e a combinação de suas previsões leva a um resultado mais geral e confiável.

3.2.4 *Ensemble learning*

Gérdon (2018) destaca que, no ensemble learning, a fusão de previsões provenientes de diversas fontes normalmente conduz a uma acurácia superior. Esta abordagem se beneficia da capacidade de cada componente em atenuar as limitações dos outros. ~~Conhecida como ensemble~~, essa técnica envolve o uso de múltiplos algoritmos ou instâncias, cada um contribuindo para o resultado final. Algoritmos que adotam essa estratégia são frequentemente classificados como métodos de *ensemble.*

A Figura 3 mostra um exemplo de como essa abordagem pode ser usada. Nesse caso, três modelos são treinados em um conjunto de dados; as previsões desses modelos são então combinadas para obter um resultado mais preciso. Uma das técnicas de combinação comuns em ensemble learning é a votação, onde cada modelo contribui com sua previsão para uma amostra e a classe mais votada é escolhida como resultado final.

Figura 3—Representação do processo de *Ensemble Learning*. A figura ilustra como múltiplos modelos, nomeados aqui como “Modelo 1”, “Modelo 2” e “Modelo 3”, recebem uma entrada de dados comum. As previsões desses modelos são então combinadas para produzir um resultado unificado e mais preciso.



Fonte: Elaborado pelo autor.

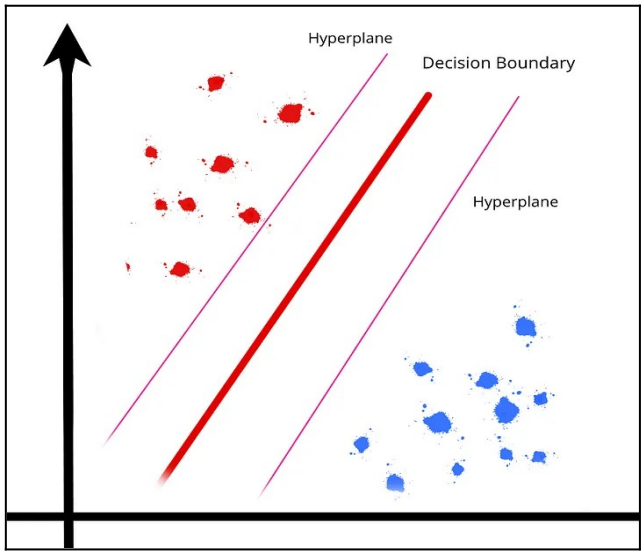
3.2.5 Máquina de vetores de suporte

Máquina de Vetores de Suporte (em inglês, *Support Vector Machine* ou SVM) é um algoritmo de aprendizado de máquina supervisionado utilizado principalmente para classificação e regressão. Ele é reconhecidamente um dos algoritmos de aprendizado de máquina mais robustos e eficazes, utilizado em uma ampla gama de aplicações, incluindo processamento de imagens, reconhecimento de fala e processamento de linguagem natural.

O algoritmo SVM busca encontrar o hiperplano que melhor separa as classes ~~em dados~~, sendo comumente usado para dois grupos, mas também adaptável para múltiplas classes. Este hiperplano, que pode ser visualizado como uma linha ou superfície, segmenta o espaço de dados de tal forma que cada lado contém apenas amostras de uma classe específica.

A essência do SVM reside em maximizar a margem, que é a distância entre esse hiperplano e os pontos mais próximos de ambas as classes, conforme ilustrado na Figura 4.

Figura 4— Representação gráfica do algoritmo SVM, exibindo o hiperplano central como a fronteira de decisão entre duas classes de dados (amostras vermelhos e azuis). Os hiperplanos adicionais, situados à esquerda e à direita do limite de decisão, definem a margem máxima entre os vetores de suporte e o hiperplano central.



Fonte: Mishra (2023).

Na Figura 4, o hiperplano (linha vermelha) representa a fronteira de decisão ideal entre as classes. Os pontos mais próximos a esta fronteira, conhecidos como

vetores de suporte, são fundamentais na definição da margem e estão destacados, representando as classes distintas.

A margem define a distância entre o hiperplano e os vetores de suporte, e o propósito principal do SVM é maximizar essa margem, garantindo uma separação robusta e generalizadora entre as classes.

4 MATERIAL E MÉTODOS

Para desenvolver uma API voltada ao diagnóstico de diabetes, recorremos a duas bases de dados específicas. Esses conjuntos de dados estão disponíveis no *National Institute of Diabetes* and *Digestive and Kidney Diseases*, vinculado ao Instituto Nacional de Saúde dos Estados Unidos.

O objetivo dessas bases de dados é prever indicativamente se um paciente tem diabetes considerando diferentes características incluídas no conjunto de dados.

4.1 BASES DE DADOS

As bases de dados foram selecionadas considerando a presença de variáveis que são fatores indicativos para o diagnóstico de diabetes. Entre eles estão: número de gestações, glicemia de jejum, pressão arterial diastólica (em mm Hg), espessura da pele do tríceps (em mm), níveis de insulina, e uma variável de resultado clínico que indica a presença ou ausência de diabetes no paciente.

A primeira base de dados, denominada *"Diabetes from Lab01"* (Mendes, 2018), contém 10 características e um total de 15.000 instâncias. A segunda base de dados, chamada "*Pima Indians Diabetes Database*" (UCI, 2016), possui 9 características e 768 instâncias. As duas bases de dados estão disponíveis no *Kaggle* compartilham informações demográficas e médicas, apresentando características semelhantes, conforme detalhado na Tabela 2.

Tabela 2: Comparação das características presentes nas bases de dados 'Base 1' e 'Base 2' utilizadas no estudo de diagnóstico de diabetes.

Característica	Descrição	(Mendes,2018)	(UCI,2016)
PatientID	Identificador do paciente.	Não	Sim
Pregnancies	Número de gestações.	Sim	Sim
Plasma Glucose	Concentração de glicose no plasma.	Sim	Sim
Diastolic BloodPressure	Pressão arterial diastólica.	Sim	Sim
Triceps Thickness	Dobra da pele do tríceps.	Sim	Sim
SerumInsulin	Insulina sérica.	Sim	Sim
BMI	Índice de massa corporal.	Sim	Sim
DiabetesPedigree	Função de linhagem de diabetes.	Sim	Sim
Age	Idade.	Sim	Sim
Diabetic	Variável de classe.	Sim	Sim

Fonte: Elaborada pelo autor.

4.2 PRÉ-PROCESSAMENTO DOS DADOS

Com o objetivo de facilitar a análise de dados, as bases de dados *Pima Indians Diabetes Database* e *Diabetes from Lab01* foram fundidas em único conjunto de dados.

No entanto, antes de realizar a junção, é necessário que ambas as tabelas tenham os mesmos nomes de colunas. As colunas do DataFrame Pima Indians Diabetes Database foram renomeadas para padronização, e a coluna ‘*PatientID*’ do *DataFrame Diabetes from Lab01* foi removida, considerando que seu conteúdo, focado na identificação individual, não contribui para a análise agregada dos dados de diabetes.

Usamos o método *rename()* do módulo *Pandas* para renomear os rótulos do *DataFrame Pima Indians Diabetes Database*, conforme ilustrado na Figura 5. O método aceita, como argumentos, um dicionário para mapear os nomes novos e um parâmetro opcional ‘*inplace*’. O argumento ‘*inplace*’, definido como *True*, assegura que a renomeação ocorra diretamente no *DataFrame* original.

Figura 5 – Código-fonte renomeando colunas do *DataFrame* 'dfPimaIndia' para formatos mais descritivos usando o método *rename()* da Pandas.

```
dfPimaIndia.rename(columns={
    'Glucose': 'PlasmaGlucose',
    'BloodPressure':'DiastolicBloodPressure',
    'SkinThickness':'TricepsThickness',
    'Insulin':'SerumInsulin',
    'DiabetesPedigreeFunction':'DiabetesPedigree',
    'Outcome':'Diabetic'
}, inplace = True)
```

Fonte: Elaborada pelo autor.

A renomeação de colunas é crucial para a junção dos *DataFrames*, pois assegura a compatibilidade dos nomes das colunas. Isso permite que a função de junção identifique corretamente as colunas correspondentes entre as tabelas.

A junção das tabelas foi realizada utilizando o método *merge()* da biblioteca *Pandas*, que combina as duas tabelas com base em colunas correspondentes. O argumento *how* do método especifica o tipo de união, como outer join, que foi utilizado para garantir a inclusão de todas as linhas, preenchendo as ausências com 'NaN' (*Not a Number*).

Figura 6 – Código-fonte utilizando Pandas para realizar a junção dos *DataFrames 'df_left'* e *'df_right'* através de um *'outer join'*, resultando em um *DataFrame* unificado "df".

```
import pandas as pd

df_left = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
df_right = pd.DataFrame({'C': [7, 8, 9], 'D': [10, 11, 12]})
df = pd.merge(df_left, df_right, how='outer')
```

Fonte: Elaborada pelo autor.

A saída do código acima é a seguinte:

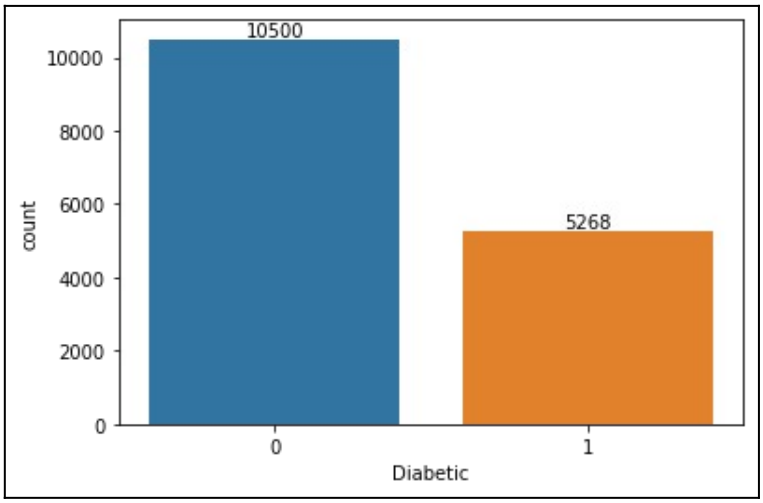
Tabela 3 – Exemplo de *DataFrame* após junção, mostrando valores nas colunas A, B, C e D com presença de dados faltantes (*NaN*).

	A	B	C	D
0	1	4	*NaN*	10
1	2	5	*NaN*	11
2	3	6	*NaN*	12
3	*NaN*	*NaN*	7	*NaN*
4	*NaN*	*NaN*	8	*NaN*
5	*NaN*	*NaN*	8	*NaN*

Fonte: Elaborada pelo autor.

O *DataFrame* resultante da fusão das duas tabelas, denominado df_diabetes, apresentou uma distribuição de 10.500 pessoas sem diabetes e 5.268 com a doença, conforme ilustrado no gráfico de barras na Figura 7.

Figura 7 – Distribuição de diabéticos (1) e não diabéticos (0) na base de dados, com 10.500 não diabéticos e 5.268 diabéticos.



Fonte: Elaborada pelo autor.

Após a fusão das tabelas, conduziu-se uma verificação para elementos faltantes (NaN) usando df_diabetes.isnull().sum(), que confirmou a ausência de dados ausentes nas colunas. Também foi realizada a remoção de entradas duplicadas, assegurando a integridade e a unicidade dos dados no conjunto final.

Durante a análise do *DataFrame* df_diabetes, observou-se um grande número de *outliers*, o que pode causar distorções no modelo. Para a remoção de *outliers*, aplicou-se o método de John Tukey (apud Datatest, 2020), que determina os limites de exclusão com base nos quartis dos dados (q1 e q3). Qualquer dado abaixo de q1 – 1,5 x (q3 – q1) ou acima de q3 + 1,5 x (q3 – q1) é considerado um *outlier* e removido. Esta abordagem ajuda a eliminar valores extremos que poderiam distorcer a análise.

É importante mencionar que, embora o método de Tukey recomende um fator de multiplicação de 1,5 para identificar e excluir *outliers*, essa medida pode não ser adequada para todas as situações. Assim, adaptamos a abordagem utilizando a biblioteca Pandas para essa exclusão de *outliers*.

4.3 ENRIQUECIMENTO DOS DADOS

Nessa seção, discute-se um processo de adição de informações relevantes ao conjunto de dados existente. No contexto da classificação de diabetes, o enriquecimento dos dados pode ser crucial para melhorar a precisão dos modelos preditivos.

O conjunto de dados foi enriquecido com informações de fontes externas, um processo desafiador, mas crucial para aprimorar a qualidade e utilidade da análise.

Para a criação de novos campos no conjunto de dados, relacionados ao Colesterol e à Fração Relevante da Hemoglobina Glicada (HbA1c), empregou-se a técnica de regressão linear. Essa abordagem estatística foi aplicada utilizando os dados disponibilizados por HOYT (2019) e RASHID (2020) como base.

4.3.1 Colesterol

O desenvolvimento do colesterol alto está fortemente ligado ao estilo, incluindo fumo, obesidade, sedentarismo, histórico familiar e alimentação (Ministério da Saúde, 2020).

Pessoas com diabetes tipo 2 também são mais propensas a ter baixos níveis de colesterol HDL e altos níveis de colesterol LDL, pois o açúcar elevado no sangue pode danificar as paredes arteriais.

Tanto o colesterol alto quanto o diabetes tipo 2 estão relacionados a um estilo de vida sedentário, dieta rica em gordura e açúcar, e sobrepeso. Assim, variável '*colesterol*' será adicionada ao conjunto de dados e utilizada como um indicador para a classificação de diabetes, considerando sua relevância clínica.

4.3.2 Fração relevante da hemoglobina glicada

Segundo Pimazoni Netto *et al* (2009), o indicador HBA1C é um marcador amplamente reconhecido e aceito pela comunidade científica desde 1993 para avaliar o controle glicêmico em pacientes diabéticos.

Em nosso estudo, assim como a variável '*colesterol*', a variável '*HBA1C'* será adicionada ao "*df_diabetes*". Esta inclusão foi baseada em uma análise de regressão

linear, que avaliou sua correlação com variáveis relevantes, como massa corporal (BMI) e idade, fortalecendo a robustez do modelo para o diagnóstico de diabetes.

4.4 DIVISÃO DA BASE DE DADOS EM CONJUNTOS DE TREINAMENTO E TESTE

A divisão correta da base de dados em conjuntos de treinamento e teste é crucial no desenvolvimento de algoritmos de aprendizado de máquinas confiáveis. O conjunto de treinamento ajusta os parâmetros do modelo, e o conjunto de teste avalia o desempenho deste em dados novos para garantir sua capacidade de generalização.

Neste estudo, a proporção utilizada para dividir os dados é de 75% para treinamento e 25% para teste. Esta divisão é considerada equilibrada, garantindo dados suficientes tanto para treinar o modelo quanto para testar seu desempenho.

Além disso, para uma avaliação mais robusta, implementou-se a estratégia de validação cruzada utilizando a função *KFold* da biblioteca *sklearn*. Este método envolve dividir os dados em múltiplos conjuntos de treinamento e teste, com cada *fold* servindo como um teste independente, aumentando a confiabilidade da avaliação do modelo.

A avaliação de desempenho de cada modelo, dentro do contexto da validação cruzada, foi realizada usando a função *cross_val_score* da *sklearn*, que calcula a precisão do modelo para cada *fold*, oferecendo uma visão abrangente do seu desempenho geral.

Esta abordagem permite uma análise aprofundada do desempenho dos modelos, levando em consideração a variação nos dados devido à divisão em conjuntos de treinamento e teste ~~e à técnica de validação cruzada~~.

4.5 CRIAÇÃO DE MODELOS DE APRENDIZADO DE MÁQUINA

Seguindo a preparação dos dados detalhada nas Seções 4.1, 4.2 e 4.3, esta seção focará na descrição e configuração específica dos modelos de aprendizado de máquinas empregadas nesta monografia.

4.5.1 *Dummy classifier*

Na avaliação dos modelos de aprendizado de máquina para o diagnóstico de diabetes, empregamos o *Dummy Classifier* como uma linha de base para comparação. Este modelo, que serve como um ponto de referência básico, foi configurado para adotar a estratégia '*most_frequent*'. Essa abordagem significa que o *Dummy Classifier* sempre prevê a classe mais frequente no conjunto de treinamento. A adoção da estratégia 'most_frequent' no Dummy Classifier estabelece um ponto de partida para a avaliação, onde modelos que superam sua eficácia demonstram capacidade de aprendizado além da simples identificação da classe mais comum nos dados.

4.5.2 Árvore de decisão (*Decision tree*)

Similarmente à subseção anterior, foi aplicada a busca em grade para obtenção dos melhores hiperparâmetros, sendo a profundidade máxima de uma árvore de decisão, representada pelo parâmetro ‘*max_depth*’,O critério de separação, especificado pelo parâmetro criterion, determina a medida pela qual a qualidade de uma divisão é avaliada em cada nó da árvore. Dois critérios comuns são o Gini e a Entropia. O Gini mede a impureza dos conjuntos resultantes após a divisão, enquanto a Entropia avalia a quantidade de informação contida nos conjuntos, O parâmetro ‘*max_features*’ que controla o número máximo de características que a árvore considerará ao procurar pela melhor divisão em cada nó e o parâmetro min_samples_leaf estabelece o número mínimo de amostras necessárias para formar uma folha (nó terminal) na árvore.

O código-fonte usado para exploração dos valores é apresentadona Figura 8. Os melhores valores obtidos para Árvores de Decisão, por sua vez, são delineados na Tabela 4.

Figura 8 – Código-fonte para otimização dos hiperparâmetros do modelo *DecisionTreeClassifier* usando *GridSearchCV*.

```
kf = KFold(n_splits=10,shuffle=True)
params= {'max_depth':[5,6,7,8,9],
         'criterion':['entropy','gini'],
         'max_features':[5,6,7,8,9],
         'min_samples_leaf':[5,6,7,8,9,10,11]
        }

arvore_test = DecisionTreeClassifier(random_state = 0)
cv = GridSearchCV(arvore_test,param_grid=params,scoring='roc_auc',cv=kf,n_jobs=-1,\
                  return_train_score=True)
cv.fit(x_treinamento,y_treinamento)
```

Fonte: Elaborada pelo autor.

Tabela 4 – Hiperparâmetros otimizados para o modelo baseado em Árvore de Decisão.

Parâmetro	Valor Selecionado
max_depth	8
criterion	'*entropy*'
max_features	9
min_samples_leaf	10

Fonte: Elaborada pelo autor.

A metodologia adotada incluiu a utilização da validação cruzada com 10 dobras (*KFold*) e embaralhamento dos dados para garantir uma avaliação robusta e imparcial do modelo.

O modelo base da Árvore de Decisão foi configurado com um estado aleatório fixo (random_state = 0) para assegurar a reprodutibilidade dos resultados.

4.5.3 Floresta randômica

Para realizar uma busca sistemática do conjunto ideal de hiperparâmetros do modelo de Floresta Randômica neste estudo, recorremos ao uso do *GridSearchCV*, uma ferramenta da biblioteca *sklearn* que realiza uma procura exaustiva pelos melhores parâmetros, melhorando a precisão e eficácia do modelo.

Ao todo, foram experimentados e cruzados 4 conjuntos distintos de hiperparâmetros:

- Conjunto 1 – Número de estimadores considerados: [26, 36, 46, 56, 66, 86, 96, 106];

- Conjunto 2 – Profundidade máxima considerada: [8, 7, 8, 9, 10, 11, 12, 13, 14, 15];
- Conjunto 3 – Número máximo de características investigadas em cada divisão da árvore: [3, 4, 5, 6, 7, 8, 9];
- Conjunto 4 – Número mínimo de amostras para gerar folha: [9, 10, 11, 12, 13];

O experimento emprega o modelo *RandomForestClassifier*, otimizado pela técnica *GridSearchCV*, seguido por uma análise detalhada do código utilizado na Figura 9.

Figura 9 – Código-fonte para otimização dos hiperparâmetros do modelo *RandomForestClassifier* usando *GridSearchCV.*

```
kf = KFold(n_splits=10,shuffle=True)
random_forest_test = RandomForestClassifier(random_state=0)
params= {
         'n_estimators':[26,36,46,56,66,86,96,106],
         'max_depth': [8,7,8,9,10,11,12,13,14,15],
         'max_features':[3,4,5,6,7,8,9],
         'min_samples_leaf': [9,10,11,12,13]
        }
cv = GridSearchCV(random_forest_test,param_grid=params,scoring='roc_auc',cv=kf,n_jobs=-1,
                  return_train_score=True)
cv.fit(x_treinamento,y_treinamento)
cv_results = pd.DataFrame(cv.cv_results_)
```

Fonte: Elaborada pelo autor.

1. random_forest_test = *RandomForestClassifier*(random_state=0): Nesta linha de código, uma instância do modelo *RandomForestClassifier* é criada, com a especificação de um estado aleatório fixo (random_state=0). Esse procedimento garante a reprodutibilidade dos resultados, contanto que todas as demais condições permaneçam constantes.
2. O dicionário '*params*' define os hiperparâmetros a serem otimizados no modelo. Os hiperparâmetros ajustados incluem:
    1. *'n_estimators'*: Quantidade de árvores na floresta aleatória.
    2. *'max_depth'*: Profundidade máxima de cada árvore.
    3. *'max_features'*: Número máximo de características consideradas em cada divisão da árvore.
    4. *'min_samples_leaf'*: Número mínimo de amostras para formar uma folha.
3. Além disso, a configuração do *GridSearchCV* envolve:

1. *random_forest_test*: O modelo RandomForestClassifier a ser otimizado.
2. *'param_grid'*: Especificação dos hiperparâmetros, conforme params.
3. *'scoring'*: Critério de avaliação do modelo, neste caso, a área sob a curva ROC (roc_auc).
4. *'cv'*: Número de dobras para validação cruzada.
5. *'n_jobs'*: Quantidade de tarefas paralelas durante a pesquisa; se None, não há paralelismo.
6. *'return_train_score'*: Se *True*, inclui a pontuação do conjunto de treinamento nos resultados.

4. cv_results = pd.*DataFrame*(cv.cv_results_): Após a conclusão da pesquisa em grade, os resultados obtidos são consolidados em um *DataFrame* do *Pandas*. Esse *DataFrame* engloba informações variadas, incluindo os diferentes hiperparâmetros testados, as pontuações obtidas na validação cruzada, as pontuações referentes ao treinamento, bem como os tempos de execução, entre outros detalhes.

A Tabela 4 exibe as métricas de avaliação do modelo de Floresta Randômica que obteve os melhores resultados com o *GridSearchCV*.

Tabela 5 – Hiperparâmetros otimizados para o modelo baseado em Floresta Randômica.

Parâmetros	Valores Selecionados
n_estimators	106
max_depth	12
max_features	3
min_samples_leaf	9

Fonte: Elaborada pelo autor.

4.5.4 Máquinas de vetores de suporte

Para compreender o desempenho do modelo de Máquina de Vetores de Suporte (SVM), avaliamos dois hiperparâmetros por meio da busca em grade. Estes são:

1. '*kernel*': Controla a função '*kernel*' utilizada pelo SVM. Foram consideradas quatro opções: 'rbf' (*Radial Basis Function*), *'poly'* (*Polynomial*), 'sigmoid', e 'linear'.

2. C: Responsável pelo parâmetro de regularização do SVM. Foram testados os valores inteiros 1, 2, 3 e 4.

A validação cruzada foi realizada com 4 dobras (*KFold*) e embaralhamento dos dados para garantir resultados robustos e imparciais (Figura 10).

O modelo base do SVM foi configurado com um estado aleatório fixo (*random_state* = 0) para garantir a reprodutibilidade dos resultados.

Figura 10 – Código-fonte para otimização dos hiperparâmetros do modelo de Máquina de Vetores de Suporte usando *GridSearchCV.*

```
params= {'kernel':['rbf','poly','sigmoid','linear'],
         'C':[x for x in range(1,5)],
        }
kf = KFold(n_splits=4,shuffle=True)
diabetic_svm =SVC(random_state = 0)
cv = GridSearchCV(diabetic_svm,param_grid=params,scoring='roc_auc',cv=kf,n_jobs=-1\
                  ,return_train_score=True)
cv.fit(x_treinamento,y_treinamento)
```

Fonte: Elaborada pelo autor.

A configuração dos hiperparâmetros da Máquina de Vetores de Suporte é apresentada na Tabela 6.

Tabela 6 – Otimização de Parâmetros em Máquinas de Vetores de Suporte com *GridSearchCV*.

Parâmetro	Valor Selecionado
C	4
kernel	rbf

Fonte: Elaborada pelo autor.

4.6 IMPLANTAÇÃO DE *SOFTWARE* PARA MODELOS DE APRENDIZADO DE MÁQUINA

Conforme destacado por Hadullo e Getuno (2021), a implantação de *software* é uma etapa crítica no ciclo de vida de desenvolvimento de software. Os métodos de implantação utilizados podem ter um impacto significativo na qualidade e na adaptabilidade do produto, o que pode afetar o sucesso do produto.

Os autores mencionam que disponibilizar previsões de modelos de aprendizado de máquinas treinados aos usuários é denominada implantação de *machine learning*. Eles também destacam que a compreensão desse processo tende a ser limitada entre cientistas de dados sem experiência em engenharia de software e que muitos engenheiros de software não são proficientes no aprendizado

de máquina. Além disso, Plonski (2019) identificou quatro métodos de implantação e analisou os requisitos, vantagens e desvantagens de cada um (Tabela 7).

Tabela 7 – Resumo das principais abordagens de implantação de modelos de Machine Learning: métodos, requisitos, vantagens, desvantagens e observações.

SN	Deployment Method	Requirements	Advantage	Disadvantages	Comment
1	Locally (Laptop or computer)	-Jupiter Notebook - R Studio or -Weka	Simple to implement Predictions on ML code	Hard to govern, monitor, scale and collaborate.	Not recommended for production
2	Hard-code the ML algorithm in the system's code.	-Jupiter Notebook - R studio or -Weka -Software program	Can be used with simple ML algorithms, like Decision Trees or Linear Regression	Hard to govern, monitor, scale and collaborate	Not recommended for production
3	Use of REST API or Web Sockets.	-Jupiter Notebook, -R studio or -Weka. -Software Framework	All requirements for the ML production system can be fulfilled.	Requires Data Scientist and Software Engineer	Recommended for production
4	Use of a commercial cloud vendor for Deployment	Colab PDE or Jupiter Notebook on Laptop	All requirements for the ML production system can be fulfilled.	Requires Data Scientist and Software Engineer	Recommended for production

Fonte: A Tabela 2, do artigo de Hadullo e Getuno (2021), ilustra a arquitetura de software de aprendizado de máquina usando o *Django REST Framework* baseado nos trabalhos de Plonski (2019).

A seleção de uma metodologia de implantação de modelos de aprendizado de máquina é uma decisão que depende de uma série de fatores. A implementação de métodos locais, como o uso do *Jupyter Notebook* ou R *Studio*, é atraente pela sua simplicidade. No entanto, essas ferramentas apresentam desafios em governança e monitoramento. Devido a estas limitações, elas não são recomendadas para ambientes de produção onde a confiabilidade e a segurança são críticas.

Por outro lado, a codificação direta de algoritmos de IA no sistema é uma abordagem viável para modelos mais simples. Essa estratégia permite um alto grau de personalização e integração. Contudo, essa prática pode não ser sustentável em uma escala maior, pois enfrenta obstáculos em termos de escalabilidade e manutenção.

Por fim, a utilização de REST APIs é frequentemente recomendada e se torna apropriada quando é possível atender integralmente aos requisitos de produção. Esta abordagem destaca a importância da colaboração entre cientistas de dados e engenheiros de software. Trabalhando juntos, eles podem criar soluções robustas que não apenas escalam eficientemente, mas também mantêm a integridade e a segurança dos sistemas de *machine learning* implementados. Essa abordagem também requer colaboração entre cientistas de dados e engenheiros de software.

Além disso, o emprego de provedores de nuvem comerciais é aconselhado quando a empresa está disposta a adotar essa abordagem e todos os requisitos de produção podem ser atendidos.

4.7 DESENVOLVIMENTO DE API COM *DJANGO* UTILIZANDO O MODELO DE FLORESTA RANDÔMICA

Neste capítulo, exploraremos a integração de um modelo aprendizado de máquina em uma API utilizando o *framework Django*.

> O *Framework Django Representational State Transfer* (*REST*) é um framework *Web* de alto nível em Python, gratuito e de código aberto, que incentiva o desenvolvimento rápido e o design limpo e pragmático. O DRF é uma ferramenta poderosa e flexível usada para construir rapidamente aplicações *Web* com base nos modelos de banco de dados do *Django*, com as seguintes vantagens: segurança, escalabilidade, customização da aplicação com serialização que suporta tanto o Mapeamento Objeto-Relacional (ORM) quanto fontes de dados não-ORM. Dado que a maioria dos modelos de aprendizado de máquina são criados usando a linguagem de programação Python, torna o DRF uma plataforma preferida para o desenvolvimento de software de aprendizado de máquina (Hadullo e Getuno, 2021, p. 152).

Django é um framework robusto para o desenvolvimento *Web* em *Python* que permite a criação de aplicativos de alto desempenho e flexíveis. Ao combinar a versatilidade do *Django* com a capacidade de previsão do modelo, seremos capazes de criar uma API que pode tomar decisões informadas com base em dados de entrada.

4.7.1 Integração da api com o modelo

A escolha do *Django* como *framework* para o desenvolvimento da API se baseia na sua capacidade de oferecer flexibilidade no desenvolvimento, a partir de uma estrutura básica, a arquitetura MTV (*model-view-template*).

A integração bem-sucedida do modelo de IA com a API do *Django Rest Framework* é essencial para a funcionalidade da aplicação. Este processo foi realizado carregando o modelo já treinado através da biblioteca *joblib*. Essa biblioteca permite que modelos treinados sejam serializados em disco e, posteriormente, deserializados para serem utilizados em outras partes da aplicação.

> A serialização de objetos é o processo de salvar um modelo de aprendizado de máquina (ML) como um Pickle, um Joblib ou manualmente, utilizando a abordagem JSON para salvar e restaurar. A serialização representa um objeto como um fluxo de bytes, a fim de armazená-lo em disco, enviá-lo por uma rede ou salvá-lo em um banco de dados. A desserialização é o processo de restaurar e recarregar o modelo de ML empacotado de volta para o formato de Notebooks Jupyter (ipynb)(Hadullo e Getuno, 2021, p. 160).

Após o carregamento (desserialização) do modelo na API, os dados de entrada fornecidos pelos usuários são submetidos ao pré-processamento.

O código da Figura 11 apresenta a definição da classe '*brain*', que encapsula a lógica de previsão de diabetes. A classe '*Brain*' está estruturada conforme as boas práticas de programação e possui dois métodos principais: '*init()*' e '*predict()*'.

Figura 11 – Código-Fonte da classe '*Brain*' para classificação de diabetes na API, mostrando os métodos de inicialização e previsão.

```
class Brain:
    def __init__(self) -> None:
        try:
            path =os.path.join('app','DiabetesPredict','brain','random_forest.joblib')
            self.randomFC = joblib.load(path)
        except Exception as e:
            print("Erro ao carregar o modelo:", str(e))
    def predict(self,exame):
        try:
            entrada_exame = list(exame.values())
            previsao = self.randomFC.predict([entrada_exame ])
            return previsao
        except Exception as e:
            print("Erro previsao", str(e))
        return None
```

Fonte: Elaborada pelo autor.

O método '*predict()*' recebe um parâmetro denominado '*exame*', o qual deve ser um dicionário contendo os valores dos atributos em questão a serem classificados. Posteriormente, o código efetua uma tentativa de previsão de diabetes com base nos dados do '*exame*', utilizando o modelo previamente carregado. Em caso de êxito na previsão, o resultado é retornado. No entanto, em situações de erro, o código gera uma mensagem de erro e retorna '*None'*.

Ademais, o código incorpora blocos '*try-except'* com o propósito de administrar situações excepcionais que possam surgir durante a execução, incluindo possíveis falhas no processo de carregamento do modelo ou erros decorrentes das operações de previsão.

Desta forma, a classe '*Brain*' encontra-se pronta para ser empregada em outras seções da aplicação, habilitada a executar previsões relacionadas à diabetes de maneira eficaz.

4.7.2 Recebimento dos dados do exame clínico e integração com modelo de *machine learning*

Antes da persistência dos dados clínicos recebidos pela API em uma base de dados, são realizados os seguintes passos críticos para a integração eficiente do modelo de ML.

1. Os dados recebidos pela API através de uma solicitação *POST* do *HTTP* são encaminhados para a classe '*ExameAPIView*', como demonstrado na Figura 12. Dentro desta classe, os dados são acessados pelo parâmetro '*request.data'* do método '*create*'. Esse procedimento facilita o tratamento de informações no formato JSON, um padrão amplamente adotado em APIs *RESTful*, permitindo um processamento eficiente e padronizado dos dados de entrada.

Figura 12 – Código-fonte da implementação da classe '*ExameAPIView*' no Django Rest Framework, apresentando o método *'create'* que processa dados de entrada recebidos via método *POST* do *HTTP*.

```
42  class ExameAPIView(generics.ListCreateAPIView):
43      queryset = Exame.objects.all()
44      serializer_class = ExamesSerial
45
46      def create(self, request, *args, **kwargs):
47          dados = request.data
```

Fonte: Elaborada pelo autor.

2. Na etapa de mapeamento dos dados, os elementos recebidos pela solicitação são organizados em um dicionário chamado *'avaliação'* (Figura 13). As chaves deste dicionário correspondem aos atributos clínicos do paciente, incluindo, mas não limitado a, '*Pregnancies'* e '*PlasmaGlucose*'. Os valores associados a estas chaves são derivados do '*request.data*', sendo convertidos para os formatos de dados correspondentes, como números inteiros e de ponto flutuante, para garantir a precisão e a correta manipulação dos dados.

Figura 13 – Trecho do código-fonte da ExameAPIView detalhando o processo de mapeamento dos dados clínicos recebidos via *POST* do *HTTP* para um dicionário no Django, com conversão de tipos para cada atributo.

```
46      def create(self, request, *args, **kwargs):
47          dados = request.data
48          avaliacao = {
49              "Pregnancies":int(dados['Pregnancies']),
50              "PlasmaGlucose":float(dados['PlasmaGlucose']),
51              "DiastolicBloodPressure":float(dados['DiastolicBloodPressure']),
52              "TricepsThickness":float(dados['TricepsThickness']),
53              "SerumInsulin": float(dados['SerumInsulin']),
54              "BMI": float(dados['BMI']),
55              "DiabetesPedigree":float(dados['DiabetesPedigree']),
56              "Age": int(dados['Age']),
57              "Colesterol":float(dados['Colesterol']),
58              "HbA1c": float(dados['HbA1c']),
59          }
```

Fonte: Elaborada pelo autor.

3. A integração efetiva do modelo de aprendizado de máquina na API é realizada através da classe '*brain*', que desempenha a função de aplicar o modelo treinado para avaliar a condição de diabetes dos pacientes. Esse passo é crucial, pois habilita a aplicação a processar e interpretar dados clínicos para fornecer previsões confiáveis.
4. Posteriormente, a realização da previsão é feita através da chamada do método '*predict()*', utilizando os dados mapeados no dicionário '*avaliação*'. Este passo é vital para a obtenção de resultados a partir do modelo. O resultado da análise é armazenado em uma variável chamada '*previsão*', que será usada para informar o usuário sobre a avaliação de risco de diabetes.
5. Além disso, há uma etapa de adição dos resultados à solicitação inicial. É verificado se o resultado da previsão é diferente de '*None*', o que indicaria uma previsão válida. Se confirmada a validade, o resultado é anexado à chave "*Diabetic*", definindo à resposta que será entregue ao usuário final.
6. Finalizando o processo, a chamada '*super().create(request, *args, **kwargs)*', é executada, finalizando a criação do objeto dentro do sistema. Este comando não só salva os dados coletados e processados no banco de dados, como também gera e envia uma resposta HTTP ao cliente. Esta resposta representa a conclusão do ciclo de solicitação e é o que o cliente recebe após a interação com a API.

Figura 14 – Segmento de código da '*ExameAPIView*' no *Django*, mostrando o uso da classe '*Brain*' para prever diabetes e o processamento subsequente da resposta, incluindo a geração de uma resposta HTTP condicional com base no resultado da previsão.

```
62            rf = Brain()
63            previsao = rf.predict(avaliacao)
64            if previsao is not None:
65                data = request.data.copy()
66                data['Diabetic'] = int(previsao[0])
67                adjusted_request = Request(request._request, data=data)  # type: ignore
68                response = super().create(adjusted_request, *args, **kwargs)
69                return response
70
71            else:
72                return Response({'error': 'Previsão de diabetes não foi bem-sucedida.'}, status=status.HTTP_400_BAD_REQUEST)
```

Fonte: Elaborada pelo autor.

Em síntese, este fluxo de trabalho exemplifica a integração de um modelo de aprendizado de máquina em uma aplicação baseada no *Django Rest Framework*. Essa aplicação lida com informações médicas, possibilitando previsões relacionadas à diabetes com base nas características clínicas dos pacientes.

4.7.3 Testes da api: avaliação da funcionalidade e precisão das previsões de diabetes

Nesta seção, apresenta-se uma visão detalhada dos testes conduzidos na API desenvolvida utilizando o *Django Rest Framework* (DRF) para classificação de diabetes. O código foi concebido para assegurar tanto a funcionalidade adequada da API quanto a precisão do modelo de previsão.

1. Para a configuração do ambiente de teste, estabeleceu-se um ambiente específico que inclui a criação de um perfil de usuário (Figura 15). Isso ocorre porque a API foi projetada para aceitar dados submetidos via método *POST* por usuários autenticados. Além disso, um objeto '*paciente*' fictício foi gerado no método *setup()* para simular cenários de teste. Um cliente de API '*(APIClient)*' foi implementado para efetuar requisições HTTP simuladas, e este cliente foi devidamente autenticado usando as credenciais do usuário de teste. Para completar a configuração, definiu-se a URL '*exames*' que permite a visualização e o teste dos dados submetidos, validando a funcionalidade da API.

Figura 15 – Exemplo do código do método *setup()* para testes na API, mostrando a criação de um usuário de teste e um objeto '*paciente*', a autenticação no cliente da API e a preparação do ambiente de teste, incluindo o carregamento do modelo de aprendizado de máquina..

```
class ExameAPITest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='anderson', password='###')
        self.paciente = Paciente.objects.create(
            name_paciente="PacienteTest",
            cpf="12345678901"
        )
        self.client = APIClient()
        self.client.force_authenticate(user=self.user)
        self.url = reverse('exames')
        self.brain = Brain()#Carga do modelo de Machine Learning
```

Fonte: Elaborada pelo autor.

2. No método '*test_criar_exame_com_previsao_diabetes()'*, exemplificado na Figura 16, foi inicializado um objeto que representa um paciente diagnosticado com diabetes. Estes dados foram submetidos à API por meio de uma solicitação *POST* direcionada à URL correspondente. A integridade da requisição foi assegurada pela validação da resposta *HTTP*, que deveria retornar o status '201 *Created'*, indicativo de criação bem-sucedida. Além disso, procedeu-se à verificação no banco de dados para confirmar que o registro do '*exame'* efetuado refletia um valor 1 no campo '*Diabetic*', sinalizando assim a detecção de diabetes pelo modelo.

Figura 16 – Código-fonte de teste na API que submete dados de um paciente hipotético com indicadores de diabetes. Nisso, valida-se tanto a criação bem-sucedida do exame no banco de dados quanto a correta classificação de diabetes pelo modelo, verificando o retorno do status '201 *Created'* e a marcação do campo '*Diabetic'* como 1.

```
def test_criar_exame_com_previsao_diabetes(self):

    dados = {
        "Paciente": self.paciente.Id,
        "Pregnancies": 3,
        "PlasmaGlucose": 102.0,
        "DiastolicBloodPressure": 100.0,
        "TricepsThickness": 25.0,
        "SerumInsulin": 289.0,
        "BMI": 42.19,
        "DiabetesPedigree": 0.18,
        "Age": 43,
        "Colesterol": 218.46,
        "HbA1c": 9.04,
    }

    response = self.client.post(self.url, data=dados, format='json')
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    exame_criado = Exame.objects.get(Paciente=self.paciente.Id)
    self.assertEqual(exame_criado.Diabetic, 1)
```

Fonte: Elaborada pelo autor.

3. No método ‘*test_criar_exame_sem_previsao_diabetes()’*, apresentado na Figura 17 e seguindo a metodologia do teste anterior, foram criados dados que simulam um paciente sem diagnóstico de diabetes. Estes dados foram submetidos à API através de uma solicitação *POST* direcionada à respectiva *URL*, e a resposta *HTTP* recebida foi verificada para confirmar o status ‘*201 Created*’, determinando a criação do registro. Adicionalmente, foi realizada uma consulta ao banco de dados para verificar se o exame correspondente ao ‘*paciente*’ tinha o campo ‘*Diabetic’* estabelecido como 0, o que indicaria a ausência de diabetes conforme esperado pelo teste.

Figura 17 –Trecho de código do teste na API que simula a submissão de um exame para um ‘*paciente*’ sem diabetes. O teste valida se a solicitação *POST* resulta no status HTTP ‘*201 Created’* e confirma que os dados persistidos no banco de dados refletem a ausência de diabetes, verificando se o campo '*Diabetic*' está definido como 0.

```
def test_criar_exame_sem_previsao_diabetes(self):
    dados = {
        "Paciente": self.paciente.Id,
        "Pregnancies": 0,
        "PlasmaGlucose": 98.0,
        "DiastolicBloodPressure": 80.0,
        "TricepsThickness": 34.0,
        "SerumInsulin": 23.0,
        "BMI": 43.50,
        "DiabetesPedigree": 1.20,
        "Age": 21,
        "Colesterol": 192.0,
        "HbA1c": 3.30,
    }

    response = self.client.post(self.url, data=dados, format='json')
    if response.status_code != status.HTTP_201_CREATED:
        print(response.content.decode())
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    exame_criado = Exame.objects.get(Paciente=self.paciente.Id)
    print(f"{exame_criado.Pregnancies}")

    exame_criado = Exame.objects.first()
    self.assertEqual(exame_criado.Diabetic, 0) # type: ignore
```

Fonte: Elaborada pelo autor.

4. Finalmente, no método *tearDown()*, o objeto de teste criado anteriormente é excluído para manter o banco de dados limpo.

5 RESULTADOS E DISCUSSÃO

No presente estudo, apresentamos uma API que objetiva fornecer um diagnóstico mais preciso de diabetes, utilizando modelos de aprendizado de máquina. Para avaliar o desempenho desses modelos, realizou-se uma avaliação individual e uma comparação abrangente entre todos os algoritmos mencionados nesta pesquisa.

5.1 DESEMPENHO INDIVIDUAL DOS MODELOS

Nesta seção será abordado o desempenho de cada modelo apresentado no estudo.

5.1.1 *Dummy classifier*

O *Dummy Classifier*, utilizado como ponto de referência inicial, registrou uma acurácia de 72%. Essa métrica reflete a prevalência da classe majoritária, que nesse contexto são os casos não diabéticos, dentro do conjunto de dados. No entanto, essa acurácia não indica uma validade preditiva autêntica, pois o modelo não tem a capacidade de diferenciar eficazmente entre as classes.

5.1.2 Floresta randômica

O modelo de Floresta Randômica, afinado com os hiperparâmetros definidos na Seção 4.5.2, emergiu com uma impressionante acurácia média de 91,74%.

Este resultado notável não é apenas um testemunho do robusto ajuste de hiperparâmetros, mas também um indicativo da habilidade deste modelo em entregar previsões precisas.

5.1.3 ÁRVORE DE DECISÃO (DECISION TREE)

Após a otimização criteriosa dos hiperparâmetros, o modelo de Árvore de Decisão se destacou com uma acurácia média de 90,08%, demonstrando sua eficácia na geração de previsões precisas.

5.1.4 Máquinas de vetores de suporte

Através da meticulosa otimização dos hiperparâmetros, detalhada na Seção 4.5.4, o modelo Máquinas de Vetores de Suporte (SVM) demonstrou uma acurácia média promissora de 87,34%.

5.2 COMPARAÇÃO DE DESEMPENHO ENTRE MODELOS

A comparação de desempenho entre modelos de *machine learning* é um elemento fundamental para a seleção do melhor modelo em uma determinada tarefa. Neste contexto, foram avaliados três modelos distintos: máquina de vetores de suporte (SVM), árvore de decisão e floresta randômica. As métricas de avaliação de desempenho, incluindo precisão, *recall* e *f1-score*, bem como a acurácia, foram utilizadas para comparar esses modelos. Os resultados da avaliação são apresentados na Tabela 8.

Tabela 8 – Resultados da avaliação de desempenho dos modelos.

Modelo	Não diabético 0 Precisão	Diabético 1 Precisão	Não diabético 0 revocação	Diabético 1 revocação	Não diabético 0 Score	Diabético 1 Score	Acurácia
Floresta Randômica	**0.94**	**0.87**	0.95	**0.85**	**0.95**	**0.86**	**0.92**
Árvore de Decisão	**0.94**	0.80	0.92	**0.85**	0.93	0.82	0.90
SVM	0.90	0.83	0.92	0.80	0.91	0.81	0.88

Fonte: Elaborada pelo autor.

A análise dos resultados revela que o modelo Floresta Randômica obteve o desempenho mais elevado em termos de acurácia, com um valor de 92%. Além disso, apresentou as melhores métricas de precisão, revocação e melhor *f1-score*

para ambas as classes, o que sugere uma capacidade sólida de classificação para o conjunto de dados em questão.

Por sua vez, o modelo Árvore de Decisão apresentou uma performance positiva, com uma acurácia de 90% e um equilíbrio entre precisão e revocação para ambas as classes, demonstrando uma habilidade consistente na classificação.

Em contrapartida, o modelo SVM mostrou um desempenho ligeiramente inferior, com uma acurácia de 88% e métricas mais baixas, em comparação com os outros modelos.

É importante notar que todos os modelos superaram o *Dummy Classifier*, que teve acurácia de 72,11%, evidenciando a eficácia do aprendizado de máquina em comparação a métodos de classificação mais simples.

Dadas essas avaliações, a escolha da API em *Django*, utilizando o modelo Floresta Randômica, é justificada pelos resultados favoráveis de desempenho.

6 CONCLUSÕES

Este estudo objetivou avaliar a eficácia da API de diagnóstico de diabetes utilizando aprendizado de máquina e discutir os desafios associados à sua implementação. Os resultados indicaram que o modelo baseado em Floresta Randômica obteve o melhor desempenho, alcançando uma precisão média de 92%. Tais resultados estão alinhados com pesquisas anteriores sobre a eficácia de modelos de aprendizado de máquina em diagnosticar diabetes.

Além disso, abordamos os desafios na implementação desses modelos, salientando a necessidade de colaboração interdisciplinar. A experiência combinada de cientistas de dados e engenheiros de *software* é fundamental para superar obstáculos técnicos e integrar eficazmente esses modelos em sistemas de saúde existentes. Essa colaboração é vital não só para a seleção de metodologias de implantação adequadas, mas também para garantir a realização de testes rigorosos e o monitoramento contínuo das previsões para assegurar sua precisão e confiabilidade.

A integração do aprendizado de máquina com o *framework Django* possibilita que as organizações superem uma variedade de desafios relacionados a estas tecnologias inovadoras, tais como gerenciamento de dados, integração de modelos com aplicações *Web,* manutenção.

Com base nos resultados e nos desafios identificados, propomos as seguintes recomendações para implementações:

1. É essencial adotar uma arquitetura de implantação robusta para projetos de aprendizado de máquina, conforme proposto por Hadullo e Getuno (2021). A personalização desta arquitetura às necessidades específicas de cada projeto pode abranger a integração de sistemas de monitoramento contínuo e a implementação de protocolos rigorosos de segurança de dados. Esta abordagem não só assegura a solidez técnica do projeto, mas também fortalece sua capacidade de resposta a desafios emergentes e requisitos variáveis.
2. A precisão e confiabilidade das previsões geradas pelos modelos de IA devem ser submetidas a uma análise rigorosa. Testes abrangentes, que examinam o desempenho do modelo em diferentes conjuntos de dados e em diversos cenários clínicos, são cruciais para validar a eficácia das

previsões. Essa prática garante que o modelo não só atenda aos padrões técnicos, mas também se alinhe às necessidades reais dos usuários finais, particularmente em contextos clínicos.

3. O monitoramento contínuo das previsões é outro aspecto fundamental para o sucesso de qualquer implantação envolvendo IA. Um sistema eficiente de monitoramento permite identificar e corrigir rapidamente quaisquer desvios ou falhas nas previsões do modelo. Essa vigilância constante é essencial para manter a confiabilidade do sistema, assegurando que os modelos continuem a operar com a precisão esperada, mesmo diante de novos dados ou mudanças nas tendências existentes.

Os resultados dessa monografia corroboram pesquisas anteriores, como o estudo de Aftab *et al*. (2021), reforçando a eficácia de modelos de aprendizado de máquina, em especial o Random Forest, para diagnóstico e gestão do diabetes. No entanto, enfatiza-se a importância de uma abordagem cuidadosa ao integrar essas tecnologias em ambientes clínicos reais, considerando não apenas a precisão dos modelos, mas também questões de segurança de dados e integração eficaz com sistemas existentes.

6.1 Perspectivas Futuras

Este estudo avalia o uso de aprendizado de máquina para auxiliar na tomada de decisão no diagnóstico de diabetes. Sua importância está ligada à busca contínua por melhorias nos métodos de diagnóstico para a diabetes, uma questão global de saúde significativa.

Pesquisas futuras podem explorar os seguintes tópicos:

1. A eficácia da API merece uma avaliação criteriosa em contextos mais amplos e diversificados. Esta avaliação deve focar na aplicabilidade da API em uma variedade de cenários clínicos e demográficos, complementada pelo *feedback* médico. A obtenção de opiniões dos profissionais de saúde sobre a ferramenta de tomada de decisão é fundamental para confirmar sua precisão e relevância clínica.
2. A pesquisa e exploração de novos modelos de aprendizado de máquina representam um caminho promissor para aprimorar a precisão e

confiabilidade na classificação de diabetes. O desenvolvimento de tais modelos pode resultar em avanços significativos nas ferramentas diagnósticas, abrindo novas possibilidades para o manejo eficaz do diabetes.

3. Testar a escalabilidade da API em situações de alta demanda é um aspecto crucial para garantir sua eficiência. É importante que a API seja capaz de processar um grande volume de dados sem comprometer o desempenho, especialmente em cenários onde decisões rápidas e precisas são necessárias.
4. A integração de tecnologias avançadas de monitoramento da glicose com algoritmos de aprendizado de máquina para a identificação precoce do diabetes é uma área de grande interesse. A combinação dessas tecnologias com métodos diagnósticos integrados oferece uma oportunidade valiosa para melhorar a detecção e o gerenciamento do diabetes.

Essas ferramentas aprimoradas têm o potencial de serem mais precisas e confiáveis, resultando em diagnósticos mais rápidos e tratamentos mais eficazes.

REFERÊNCIAS

BIG HOPES FOR BIG DATA. Nature Medicine, New York, v. 26, n. 1, p. 202-205, jan. 2020. Disponível em: <https://www.nature.com/articles/s41591-019-0740-8>. Acesso em: 13 Set. 2023.

SOCIEDADE BRASILEIRA DE DIABETES. Diretrizes da Sociedade Brasileira de Diabetes 2019-2020. Salvador: Sociedade Brasileira de Diabetes, 2019. Disponível em: <https://www.saude.ba.gov.br/wp-content/uploads/2020/02/Diretrizes-Sociedade-Brasileira-de-Diabetes-2019-2020.pdf\>. Acesso em: 13 Set. 2023.

AFTAB, S., Alanazi, S., Ahmad, M., Khan, M. A., Fatima, A., & Elmitwally, N. S. (2021). Sistema de Suporte à Decisão para Diabetes Baseado em Nuvem Utilizando Fusão de Aprendizado de Máquina. Computers, Materials & Continua. DOI:10.32604/cmc.2021.016814. Disponível em: <https://www.techscience.com/cmc/v68n1/41867/html\>. Acesso em: 13 Set. 2023.

CUBILLOS, G., Monckeberg, M., Plaza, A., *et al*. Development of machine learning models to predict gestational diabetes risk in the first half of pregnancy. BMC Pregnancy Childbirth, 23(1), 469 (2023). Disponível em: <https://doi.org/10.1186/s12884-023-05766-4>.
Acesso em:13 Set. 2023.

OLGA, Arthur Quintella de Mello *et al*. MLOps - Transformando Teoria em Prática. Disponível em: <http://repositorio.insper.edu.br/handle/11224/3723>.
Acesso em:13 Set. 2023.

HADULLO, K. O., & Getuno, D. M. (2021). Machine Learning Software Architecture and Model Workflow. A Case of Django REST Framework. Disponível em: <https://pdfs.semanticscholar.org/1d69/1e208a0b26f14247442c0253b91254381844.pdf> Acesso em:13 Set. 2023.

RED HAT. What are Application Programming Interfaces? 2021. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces#:~:text=API%20%C3%A9%20a%20sigla%20emcria%C3%A7%C3%A3o%20de%20aplica%C3%A7%C3%B5es%20de%20software>. Acesso em: 16 dez. 22.

MICHELAZZO, P. A documentação de software, 2006. Disponível em: <http://www.michelazzo.com.br/dicas-para-documentacao-de-softwares>.
Acesso em: 17 Dez. 2022.

BECKENKAMP, M. API REST e os verbos HTTP, 2016. Disponível em: <https://blog.mbeck.com.br/api-REST-e-os-verbos-http-46e189085e21>. Acesso em: 18 Dez. 2022.

GOULARTE, R.; MOREIRA, E. S. Hipermídia: problemas atuais, novas tecnologias e sua relação com empresas. Relatório Técnico n. 114. São Carlos: ICMC, 2000. Disponível em: <https://*Web*.icmc.usp.br/SCATUSU/RT/Relatorios_Tecnicos/BIBLIOTECA_113_RT_114.pdf>Acesso em: 17 Dez. 2022.

FIELDING, Roy Thomas. Chapter 5: Representational State Transfer (REST). 2000. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/REST_arch_style.htm. Acesso em: 23 Dez. 2022.

MOZILLA DEVELOPER NETWORK. O que é *Django*?. Mozilla Developer Network, 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/*Django*/Introduction >. Acesso em: 22 Out. 2023.

AUGUSTO, D. Frameworks opinativos vs não opinativo. Dev.to, 2021. Disponível em: <https://dev.to/danielaugusto/frameworks-opinativos-vs-nao-opinativo-2c4f>. Acesso em:25 mar. 2023.

SILVA, Diandra Andrade e. Como funciona a arquitetura MTV (*Django*). 2022. Disponível em: <https://diandrasilva.medium.com/como-funciona-a-arquitetura-mtv-*Django*-86af916f1f63>. Acesso em: 23 Dez. 2022

RUSSELL, S. J.; NORVIG, P. Inteligência Artificial: uma abordagem moderna. 4. ed. Rio de Janeiro: LTC, 2022.

BREIMAN, L., & Cutler, A. (2001). Random Forests. Machine Learning, 45(1), 5-32. <https://doi.org/10.1023/A:1010933404324>.Acesso em: 13 Set. 2023.

GÉRDON, Aurélien. Mãos à obra: aprendizado de máquina com Scikit-Learn e TensorFlow. Rio de Janeiro: Alta Books, 2019. 576 p.

MENDES, Fernanda. Diabetes. 2018. Disponível em: <https://www.kaggle.com/fmendes/diabetes-from-dat263x-lab01>. Acesso em: 08 Ago. 2021.

UCI MACHINE LEARNING REPOSITORY. *Pima Indians Diabetes Database*. 2016. Disponível em: <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database/data>. Acesso em: 08 Ago. 2021.

DATATEST. (2020). Como detectar *outliers*. Disponível em: <https://datatest.readthedocs.io/en/stable/how-to/outliers.htm>. Acesso em: 20 ago. 2021.

SACRAMENTO, G. (2023). Árvores de decisão para classificação de clientes. Blog Somostera. Disponível em: <https://blog.somostera.com/data-science/arvores-de-decisao>.Acesso em: 13 Set. 2023

MISHRA, S. (2023). Breaking down the support vector machine (SVM) algorithm. Towards Data Science. Disponível em: <https://towardsdatascience.com/breaking-down-the-support-vector-machine-svm-algorithm-d2c030d58d42>. Acesso em: 20 Out. 2023.

HOYT, Robert, MD. Diabetes Prediction Dataset - Bioestatística Program DataSets da Universidade de Vanderbilt. Publicado em: 02 de maio de 2019. Disponível em: <https://data.world/informatics-edu/diabetes-prediction/workspace/project-summary?agentid=informatics-edu&datasetid=diabetes-prediction>. Acesso em: 02 Fev 2023.

RASHID, Ahlam. Construção do Conjunto de Dados sobre Diabetes. Publicado em: 18 de julho de 2020. Versão 1. DOI: 10.17632/wj9rwkp9c2.1. Mendeley Data da Elsevier. Disponível em: <https://data.mendeley.com/datasets/wj9rwkp9c2/1>. Acesso em: 04 Fev 2023.

MINISTÉRIO DA SAÚDE. (2020, 8 de agosto). Dia Nacional de Prevenção e Controle do Colesterol. Disponível em: <https://bvsms.saude.gov.br/08-8-dia-nacional-de-prevencao-e-controle-do-colesterol/>. Acesso em: 23 ago. 23.

NETTO, Augusto Pimazoni et al. Atualização sobre hemoglobina glicada (HbA1C) para avaliação do controle glicêmico e para o diagnóstico do diabetes: aspectos clínicos e laboratoriais. Jornal Brasileiro de Patologia e Medicina Laboratorial, v. 45, n. 1, p. 31-48, fevereiro 2009. Disponível em: <https://doi.org/10.1590/S1676-24442009000100007> Publicado em 20/02/09. Acesso em: 17 Nov. 2023.

PLONSKI, P. Deploy Machine Learning Models with Django. Disponível em: <https://www.deploymachinelearning.com/start-django-project/>. Acesso em: 13 Ago. 2023.