Técnicas de Aprendizado de Máquina na Previsão do Câncer de Mama

Vinícius Pacheco Vieira[1], **Mirkos Ortiz Martins**[1]

[1]Curso de Ciência da Computação - Centro Universitário Franciscano

vinipachecov@gmail.com, mirkos@unifra.br

Abstract. *This paper describes the development of a machine learning application by using Python programming language and it's assets for pattern matching a group of classified breast cancer data base divided between malignant and benign. The objetive of the algorithm is the classification of the tumor data and allow the insertion of a new tumor data between the classified ones and predict the diagnosis with the application. It was used a deep neural network from the deep learning technique with Google's framework Tensorflow and a Support Vector Machine(SVM) from scikit-learn library. The final accuracy of SVM the and the deep neural network was respectively, 97% and 0.96% for a database with 398 diagnosis used for the training dataset and 171 for test dataset. The dataset used was provided by the Wisconsin University from real tumors, which allows us to conclude that the software can help pathologysts for better diagnosis of malignant breast cancer tumors.*

Resumo. *Este trabalho descreve o desenvolvimento de uma aplicação de machine learning ou data science, utilizando a linguagem de programação Python, para o reconhecimento de padrões em um conjunto de diagnósticos de câncer de mama classificados entre benignos e malignos. O objetivo do algoritmo é a classificação entre os diagnósticos e permitir a inserção de um novo exame dentre os dados classificados e então inferir o diagnóstico com a aplicação. Foi utilizada a técnica de redes neurais em aprendizado profundo e máquinas de vetores de suporte e o resultado alcançado foi respectivameente de 97% e 96% de eficácia, para uma base de 398 diagnósticos usados como treinamento para o algoritmo e 171 diagnósticos para a fase de classificação - fornecidos por uma base real de diagnósticos - permitindo afirmar que o uso dessa ferramenta pode auxiliar no trabalho de classificação de tumores de câncer de mama entre as categorias benignas e malignas.*

1. Introdução

O câncer de mama é o tipo que mais atinge as mulheres segundo [da Silva]. Segundo o estudo, cerca de 28% dos novos casos de câncer todos os anos correspondem a exemplares de câncer de mama. Em 2013, o número de mortes relacionadas ao câncer de mama correspondeu a 14.388, sendo 181 homens e 14.206 mulheres ainda segundo [da Silva].

Essa doença, atinge geralmente públicos na faixa de 35 anos ou mais, evoluindo suas chances de ocorrência gradualmente com o avanço da idade. Com o envelhecimento que estamos presenciando nas sociedades modernas, realizar pesquisas para auxiliar no

diagnóstico precoce de doenças como o câncer de mama, mostram-se investimentos fundamentais para a saúde pública.

Diante desse cenário, muito se tem pesquisado como introduzir métodos para aumentar a eficácia e diagnosticar mais precocemente a doença. Estudos como [Melo et al. 2014] mostram como algoritmos de aprendizado de máquina podem auxiliar no diagnóstico do câncer de mama atrávez da análise de mamografias. As taxas de acurácia chegaram a 80% em alguns algoritmos como em redes neurais artificiais de arquitetura Multi-Layer-Perceptron (MLP). Com base neste domínio e sabendo da performance do algortimo de redes neurais utilizado em [Melo et al. 2014], este trabalho pretende comparar algumas técnicas de aprendizado de máquina na avaliação de dados do câncer de mama disponibilizados pela Universidade de Wisconsin [1]. Por meio desta análise será comparada as implementações e seus resultados relacionados ao tipo de técnica utilizada de aprendizado de máquina.

2. Objetivos

O objetivo geral deste trabalho foi comparar as diferentes implementações e performances dos algoritmos de aprendizado de máquina na classificação de dados de amostras do câncer de mama. Os algoritmos utilizados foram: redes neurais artificiais *Multi-Layer-Perceptron*, redes neurais profundas com o framework *Tensorflow* e *Support Vector Machines* com a implementação da biblioteca *Scikit-Learn*.

2.1. Objetivos Específicos

Dentre os objetivos específicos desse trabalho, foram identificados:

- Estudar sobre Redes Neurais, principalmente as de modelo Multi-Layer Perceptron, para identificar a melhor abordagem para a classificação dos dados;
- Construir um modelo de Rede Neural para classificação dos tumores;
- Fazer uso da Linguagem Python, juntamente da API do IBM-Watson, para construção do modelo de classificação ou como alternativa, Python com o framework TensorFlow da Google, como base para construção do modelo de classificação;
- Fazer uso dos dados de amostras do câncer de mama disponibilizados pela Universidade de Wisconsin;
- Um software que tem a capacidade de classificar amostras de câncer de mama com base em variáveis de entrada a respeito do tumor;

3. Referencial teórico

Para a construção de uma aplicação que auxilie na predição de diagnóstico de câncer de mama, conforme explicado na introdução desse trabalho, foi proposta uma abordagem na área de inteligência artificial, mais especificamente em *machine learning* os quais serão apresentados a seguir. Também será explicado como estão estruturadas as informações dos exames laboratoriais usados como base de treinamento para o software desenvolvido.

3.1. Inteligência Artificial

EXPLICAR O QUE É [referenciado]

[1] Base de dados de diagnósticos de câncer de mama: `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\%28Diagnostic\%29`

3.2. Classificação de Dados

O campo de machine learning, que é uma área da inteligência artificial, engloba um conjunto de problemas definidos como classificação de dados. Este trabalho seguiu uma métrica para atingir as metas e os objetivos do descritos na introdução, organizado conforme o fluxograma mostrado na Figura 1, com os passos previamente elaborados e seguidos durante a implementação.

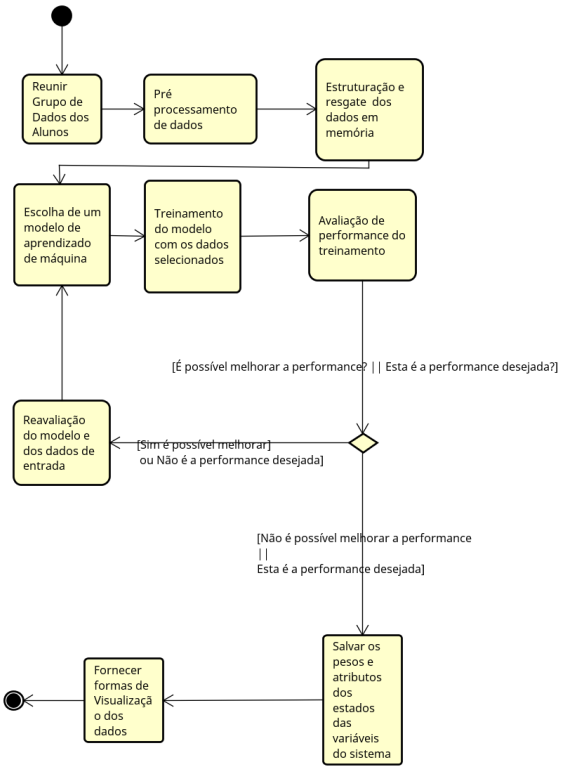

Figura 1. Sequência de processos para implementação de *machine learning*.

Após a coleta dos dados, na primeira etapa é feita a classificação dos dados, contigos em um arquivo em formato de texto separados por vírgula, `.csv`. A segunda etapa necessária é efetuar um pré-processamento de eliminação de valores inconsistentes, como caracteres ao invés de números para os campos de valores numéricos e espaços vazios, que devem ser preenchidos com o valor zero. Na existência de entradas de dados com variáveis incompletas, são possíveis duas escolhas: retirar o exemplar ou atribuir algum valor à variável. No caso de o *dataset* ter um tamanho pequeno, a retirada pode comprometer o equilíbrio entre as classes a serem classificadas, por conter alguma informação importante.

Nesse caso, seria possível fazer a média dos valores da coluna na qual o dado faltante encontra-se e preenchê-la com o resultado do cálculo. Outra saída possível é desfazer-se de dados faltantes sem comprometer muito o balanço entre classes, o que não impede de também se adotar a primeira opção, a de calcular a média, ou outros cálculos para preencher com o resultado.

Nesse trabalho, optou-se por ..... os dados constantes no *dataset*, quando incompletos.

3.3. Padronização ou Normalização de dados

O resultado da padronização ou normalização z-escore é quando redimensionar nossas *features* para que elas tenham como propriedades, as de uma distribuição normal com média igual a zero e desvio padrão igual a um.

A padronização de *features*, para que elas fiquem centradas em zero e com desvio padrão em 1, não é importante somente se iremos comparar medições com diferentes unidades de medida, mas, em geral, é um requisito para muitos algoritmos de *machine learning*. Se tomarmos como exemplo o algoritmo *gradient descent*, usado para minimizar as funções de custo quando as *features* estão em diferentes escalas, alguns pesos irão atualizar-se mais rapidamente que outros, pois os valores brutos do *dataset* participam na atualização dos valores dos pesos. Na Figura 2, é possível observar as fórmulas para padronizar (*Standardisation*) e normalizar (*Normalisation*) os dados.

Standardisation	Normalisation
$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation }(x)}$	$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$

Figura 2. Formas de colocar dados na mesma escala

3.4. Separação em Dataset de Treino e Testes

A separação de *datasets* também é um passo importante para otimizarmos os resultados do algoritmo de *machine learning*. O motivo dessa separação é para que, dentro do próprio *dataset*, já haja uma avalição prévia da performance do algoritmo, em que uma parcela do *dataset* ficará para treinamento (mais da metade), e o restante, para testes.

Consequentemente, o *dataset* de testes serve como entrada para a avaliação das predições baseadas no treinamento do *dataset* de treinos. Utilizar esse tipo de técnica sugere, também, que o *dataset* deve ser grande o suficiente para conter um número razoável de casos do problema a ser estudado e evitar falhas simples de generalização.

3.5. Estruturação e Resgate dos dados em Memória

Após a etapa de pré-processamento, é possível passar para a próxima etapa, que consiste no resgate dos registros do banco de dados, estejam eles em uma planilha, em um arquivo .xls, ou em linhas e separados por vírgula, como um arquivo .csv ou em um SGBD, como MYSQL ou Postgre. Com isso, é preciso utilizar uma linguagem de programação (junto ou não de um framework) ou plataforma como o Weka, para acessar os dados em memória para que sejam trabalhados. Nesse processo, é importante perceber que, geralmente, a última coluna da tabela corresponde ao resultado e, por isso, ela deve ser acessada separadamente, formando um vetor de resultados. O restante da tabela de variáveis consiste na matriz de atributos.

3.6. Escolha de um algoritmo de Aprendizado de Máquina

A escolha de um algoritmo de aprendizado de máquina [Blum 2007] deverá ser pensada de acordo com a necessidade do problema. Os problemas de aprendizado de máquina dividem-se, basicamente, entre regressão e classificação de dados.

Para compreender em qual conjunto um dado problema encontra-se, basta pensar se o problema tem um número contínuo de valores, como, por exemplo, o preço de uma ação na bolsa de valores, ou se existe um conjunto finito e discreto de valores que o nosso problema pode originar. No primeiro caso, estamos falando de problemas de regressão e, no segundo, de classificação de dados.

3.7. Treinamento do algoritmo

Para o treinamento do algoritmo ter um resultado desejado e levar a predições relevantes, é preciso que todas as etapas anteriores tenham sido completes; do contrário, os algoritmos, mesmo sendo executados, não entregarão o resultado desejado nas etapas seguintes.

Para realizar o treinamento, é necessário compreender a estrutura interna do algoritmo e como funciona o processo de aprendizado que será utilizado para tal. Cada algoritmo adota uma estratégia diferente, apesar de sempre se basear em conceitos comuns de erro ou custo geral dos *datasets*.

3.8. Avalição de Performance do Treinamento

Para que tenhamos uma orientação realista dos resultados oferecidos pelo treinamento do nosso algoritmo e seu modelo, é absolutamente necessário utilizar os indicadores estatísticos e compreendê-los plenamente antes de propor conclusões. Dentre os indicadores estatísticos mais utilizados, estão acurácia, precisão e *recall*.

Acurácia pode ser traduzida como os acertos das predições do algoritmo (no *dataset* de testes ou em novos casos), ou seja, quando as predições do algoritmo levam a uma avaliação correta do que já aconteceu ou daquilo que se verifica como verdadeiro no futuro (estimativa de probabilidade).

3.9. Classes desbalanceadas

Entretanto, utilizar apenas a acurácia como único indicador de performance não é indicado e pode dar-nos falsas impressões sobre uma alta performance. Se pegarmos um exemplo crítico, como um classificador para diagnosticar um determinado tipo de câncer, e tivermos 99% de taxa de acerto (acurácia), podemos ficar impressionados e pensar que temos um excelente resultado em mãos.

O problema é que, desses 99%, apenas 0.5% do *dataset* tinha a doença e caiu na taxa de erro. Nesse caso, acertaram-se diagnósticos onde não havia câncer, mas falhou-se onde havia. Certamente, não podemos considerar isso um bom resultado, ainda mais em se tratando de uma questão crítica, como uma indicação de probabilidade de uma doença tão grave. Esse tipo de situação pode ser melhor compreendida e mensurada por outros classificadores, como precisão e *recall*. Em outras palavras, podemos ver o problema dos diagnósticos na Tabela 1.

Tabela 1. Classificação dos dados

	Positive	**Negative**
Positive	true positive	true negative
Negative	false positive	false negative

Dentro do nosso problema da predição do câncer, podemos interpretar os elementos da tabela da seguinte forma:

- True Positive (TP): Número de exemplos positivos da doença e marcados como tal.
- False Positive (FP): Número de exemplos falsos da doença e marcados como positivos.
- True Negative (TN): Número de exemplos negativos da doença e marcados como tal.
- False Negative (FN): Número de exemplos positivos da doença e marcados como negativos.

Utilizando o problema sobre o diagnóstico de câncer, deparamo-nos com duas situações indesejadas: primeiro, o erro de classificar uma pessoa como portadora da doença, mas que, na verdade, não a contém; e segundo, o erro de não classificar uma pessoa como portadora da doença. É possível imaginar as consequências, tanto do primeiro erro, em que um paciente pode iniciar um tratamento por conta de um falso positivo, quanto no segundo caso, em que um possível paciente é diagnosticado sem a doença, mas deveria ser avaliado para iniciar um tratamento.

3.9.1. Paradoxo da acurácia

Conforme explicado por Abma (2009), a acurácia pode, em determinados casos, não ser uma fonte confiável da performance do algoritmo de *machine learning*, levando-nos ao paradoxo da acurácia.

De forma resumida, é um problema de balanceamento de classes, em que existe um número maior de *False Positives* do que de *True Positives* e em que a acurácia pode aumentar caso o classificador identifique tudo como negativo [Abma 2009]. O contrário também é válido, quando o número de True Negatives é menor do que o número de False Negatives, e o classificador identifica tudo como positivo. Sendo assim, são sugeridos outros indicadores, além da acurácia, para complementar a análise, entre eles *precision* (Equação 1) ou precisão e *recall* (Equação 2).

$$Precision = \frac{tp}{tp + fp} \tag{1}$$

$$Recall = \frac{tp}{tp + fn} \tag{2}$$

3.10. F-Score ou Escore-F

O *F-Score* ou *Escore-F*, conforme citado por[Abma 2009] e [Fawcett 2004], trata-se de uma medida calculada pela média harmônica entre precisão e *recall*. Ele é considerado um indicador que varia de 0 até 1, sendo que 0 é resultado ruim de teste, e 1, um resultado ideal. Na Equação 3, apresenta-se a fórmula do *F-Score*.

$$F = 2.\frac{precision.recall}{precision + recall} \tag{3}$$

3.11. Avaliações

No momento em que temos um algoritmo treinado e temos as métricas para avaliarmos a *performance*, devemos avaliar até que ponto é interessante chegar em um resultado dentro dos recursos disponíveis para a pesquisa.

Nas pesquisas da área de *Deep Learning* ou Redes Neurais Profundas, é comum a necessidade de máquinas diferenciadas e projetadas especificamente para esse propósito, o que não será possível neste trabalho. É evidente que, em um *software* sob medida, é possível exigir também um *hardware* determinado para que uma funcionalidade atinja o objetivo desejado; entretanto, nesta pesquisa, este quesito terá como limitação a máquina que utilizarei. Para isso, estabeleceremos um limite de treinamento, de forma intuitiva, com base na *performance* apresentada pela máquina a ser utilizada.

Dependendo de sua *performance*, é possível reavaliar e voltar para a escolha de um novo modelo de estrutura de rede neural, além de recomeçar o processo de treinamento e avalição até que seja encontrado o resultado desejado ou o mais próximo dele.

3.12. A base de dados de diagnósticos de câncer de mama

As características são calculadas a partir de uma imagem digitalizada de um aspirado com agulha fina (FNA) de uma massa de peito. Eles descrevem características dos núcleos celulares presentes na imagem. n o espaço tridimensional é o descrito em: [K. P. Bennett e O. L. Mangasarian: ”Discriminação de programação linear robusta de dois conjuntos linearmente inseparáveis”, métodos de otimização e software 1, 1992, 23-34].

Este banco de dados também está disponível através do servidor ftp UW CS: ftp ftp.cs.wisc.edu cd math-prog / cpo-dataset / machine-learn / WDBC /

Informações do Atributo:

1) ID número 2) Diagnóstico (M = maligno, B = benigno) 3-32)

Dez características de valor real são computadas para cada núcleo celular:

A média, erro padrão e ”pior”ou maior (média dos três maiores valores) dessas características foram calculadas para cada imagem, resultando em 30 recursos. Por exemplo, o campo 3 é Radius Médio, o campo 13 é Radius SE, o campo 23 é o Pior Raio.

Todos os valores das características são recodificados com quatro dígitos significativos.

Valores de atributo perdidos: nenhum

Distribuição de classe: 357 benigna, 212 maligna

3.13. Estrutura do Software

Após encontrarmos o resultado desejado ou chegarmos ao que entendemos ser o melhor resultado possível, dentro dos modelos escolhidos e das limitações de máquina que possuímos, podemos começar a pensar em funcionalidades relativas ao *software*. Dessa forma, iremos alcançar o objetivo final, integrando com funcionalidades, como interface gráfica e banco de dados, proporcionando uma melhor visualização do cenário da IES.

3.14. Trabalhos Correlatos

Trabalhos como de [Becker et al. 2017] mostram a importância e o alto grau de relevância de pesquisas na classificação de tumores utilizando as técnicas de aprendizado de máquina. No estudo, foram utilizadas redes neurais artificiais convolucionais em aprendizado profundo, chegando a conclusão que a acurácia geral de classificação de tumores do câncer de mama por mamografias era a mesma dos radiologistas.

Dentro de uma outra perspectiva, o trabalho [Wang et al. 2016] foi feito trabalho semelhante com aprendizado profundo, porém para classificação de câncer de mama metastático por meio de imagens. Os resultados do trabalho resultaram em prêmios e a impressionantes indicadores como 0.925 AUC de área abaixo da curva com base no gráfico ROC para imagens inteiras, lembrando que 1.0 é significaria que o classificador é perfeito para identificar a doença.

Ainda no estudo de [Wang et al. 2016] foi visto que patologistas da área chegavam ao resultado de 0.966 AUC e que ao se utilizarem da ferramenta, alcançaram a 0.995 AUC. Esse valor representa uma redução de 85% dos erros humanos de diagnósticos, reforçando a importância e relevância das pesquisas e ferramentas que podem auxiliar no aumento da acurácia dos diagnósticos dos patologistas.

4. Metodologia

Conforme as tecnologias do mercado mudam mais rapidamente e conforme a necessidade de construir-se um software com qualidade, novos métodos surgiram como renovação do mercado para os novos modelos de empresas e de demandas [Sommerville 2010]. Optou-se por utilizar uma metodologia de desenvolvimento solitário para a construção desse projeto, a nomear-se metodologia Cowboy.

4.1. Cowboy: Metodologia Ágil para Desenvolvedores Solo

É indiscutível que não há mais volta após a valorização da engenharia de *software* no processo de produção de um produto de software. Entretanto, as metodologias, em sua maioria, tratam-se de processos para equipes, deixando-nos a pergunta de como fazer um conjunto de ações objetivas, como em uma metodologia, e um processo voltados para um desenvolvedor sozinho. Pior ainda é imaginar que, se mesmo habilitadas, equipes fazem softwares e falham, logicamente não parece que o resultado seria muito melhor para desenvolvedores solitários.

Entretanto, conforme [Hollar 2006] revela, muitos dos softwares em escala global são escritos por programadores solitários. Não faltariam, também, exemplos bem sucedidos, como Bill Gates, Alan Turing e, por último e não menos importante, Linus Torvalds, o desenvolvedor do primeiro Kernel Linux.

4.2. Requisitos de software

Nas Figuras 6 e 7 o conjunto de Requisitos Funcionais e Não Funcionais do sistema:

RF1: Controle de bases de Dados	**Requisito Funcional**
O sistema deve permitir com que o usuário insira ou remova bases de dados em .xls para serem utilizadas como treinamento podendo ser guardadas em um banco de dados.	
Requisitos não funcionais associados	
Descrição	**Categoria**
Dificuldade: média	**Relevância:** alta

RF2: Controle de Histórico de performances	**Requisito Funcional**
O Sistema deverá permitir um registro de históricos de treinamento das redes neurais juntamente com seus pesos.	
Requisitos não funcionais associados	
Descrição	**Categoria**
RNF1**:** Os históricos de performances mantidos em tempo de execução deverão estar presentes em memória e poderão ser guardados no banco.	Disponibilidade de dados
RNF2: Os históricos de performances deverão constar a acurácia, o erro do modelo e o valor F1-score.	Especificação
Dificuldade: alta	**Relevância:** alta

RF3: Aprendizado de Máquina com Redes Neurais	**Requisito Funcional**
Uso de redes neurais Multi-Layer Perceptron para realizar predições da base de dados carregada.	
Requisitos não funcionais associados	
Descrição	**Categoria**
RNF3: O treinamento da rede neural não pode ultrapassar 2 horas.	Performance
RNF4: O sistema utilizará a função rectified linear function para ativação dos neurônios.	Especificação
RNF5: O sistema utilizará um momentum para acelerar o treinamento da rede neural.	Especificação
RNF6: O sistema deverá usar no treinamento uma função de encontro de gradientes diferente do gradient descent, podendo ser mini batch gradient descent ou stochastic gradient descent.	Especificação
Para acelerar o treinamento, deverá	Especificação

Figura 3. Requisitos Funcionais e Não Funcionais Parte I

ser usado o framework Tensorflow devido a sua capacidade de processamento em GPU.	
RNF7: A biblioteca Theano deverá ser usada para cálculos computacionais mais custosos como as multiplicações de matrizes.	Especificação
Dificuldade: alta	**Relevância:** Essencial

RF4: Tratamento de dados	**Requisito Funcional**
Uso de one hot encoding em variáveis de múltiplos estados, normalização por padronização, separação dos dados em dois sets sendo um para testes e outro para treinamento.	
Requisitos não funcionais associados	
Descrição	**Categoria**
RNF8: A normalização será feita pela fórmula de padronização de dados.	Especificação
RNF9: A separação dos sets de treino e testes deverá respeitar a proporção de 80% para treino e 20% para testes.	Especificação
Dificuldade: alta	**Relevância:** Essencial

RF5: Gerenciamento de pesos e performances dos treinamentos prévios	**Requisito Funcional**
O sistema deverá permitir o usuário selecionar, carregar, deletar e salvar pesos das redes neurais previamente treinadas que constam no banco de dados ou no histórico em memória.	
Requisitos não funcionais associados	
Descrição	**Categoria**
RNF10: Para carregar os dados, a arquitetura da rede neural atual deve ter a mesma da rede treinada para não haver erros ou discrepâncias nas inferências.	Especificação
RNF11: Ao carregar os pesos de um modelo deverão ser carregados juntamente o histórico de performance atribuído a aqueles pesos.	Especificação
Dificuldade: alta	**Relevância:** Alta

Figura 4. Requisitos Funcionais e Não Funcionais Parte II

A maioria dos requisitos não funcionais associados são do tipo de especificação devido a grande influência do processo de pré-processamento de dados. Essa etapa é fundamental para a obtenção de sucesso nos treinamentos. Por isso os requisitos tendem a diretamente ou indiretamente a se relacionar com essa categoria e essa etapa.

A Figura 5 representa a visão geral ou o fluxo esperado, no uso do protótipo, para a predição ou a inferência correta do diagnóstico dos turmores.

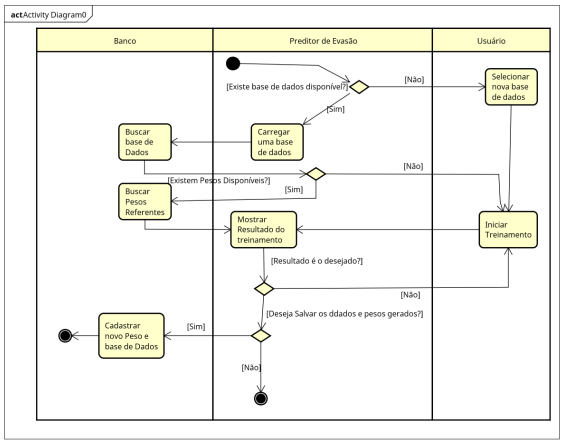

Figura 5. Diagrama de Atividades do Protótipo

A seguir, apresenta-se o Diagrama de Classes, contemplando as principais classes envolvidas na construção do *software*.

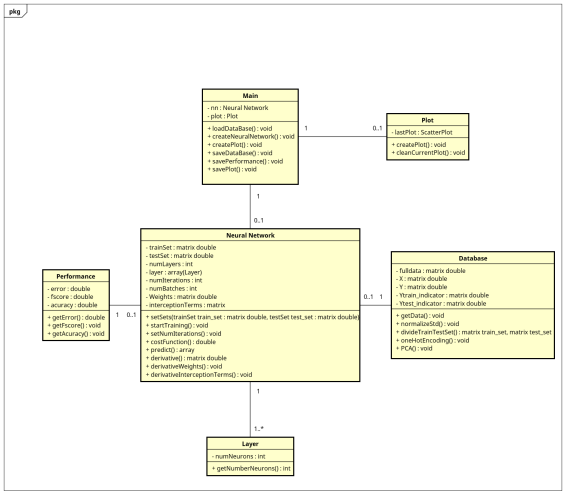

Figura 6. Diagrama de Classes

4.3. Modelo Lógico do Banco de Dados

A Figura 7 exemplifica as partes mais sensíveis do banco de dados de um possível protótipo do problema.

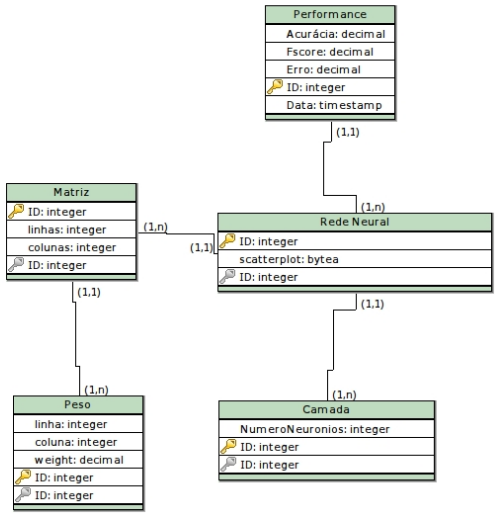

Figura 7. Diagrama de Atividades do Protótipo

O ponto mais importante a ser armazenado no banco são os dados referentes aos pesos de treinamentos das redes neurais utilizadas e sua arquitetura. Os pesos para que caso seja uma arquitetura tenha resultados eficientes em suas inferências, seja possível utilizá-los novamente para as novas inferências. A arquitetura necessita ser guardada, seja o número de camadas ou o número de neurônios por camadas pelo fato de ser necessário a utilização da mesma arquitetura para se reproduzir os pesos guardados.

5. Desenvolvimento e implementação

O protótipo a ser desenvolvido será estruturado na linguagem Python, em conjunto com o *framework* QT, para interface gráfica, e o Sistema Gerenciador de Banco de dados PostgreSQL. O projeto será orientado com base na metodologia Cowboy e em técnicas ágeis.

Python é uma linguagem de programação interpretada e de alto nível, e seu propósito é de uso geral. Conforme[Summerfield 2010], Python é uma linguagem de fácil aprendizagem e bem difundida, justamente por ser multiplataforma e por ser de fácil leitura, juntamente com sua capacidade de desenvolver aplicações com menos linhas de código do que aplicações equivalentes em C++ ou Java. Além de contar com uma biblioteca padrão muito completa, em Python, podemos programar com o paradigma procedural orientado a objeto e, até mesmo, com o paradigma funcional. Vale ressaltar que Python é uma das linguagens mais utilizadas no meio de *machine learning*, sendo, assim, a linguagem com grande quantidade de exemplos e comunidade forte para essa área que engloba este trabalho.

Devido à essência solitária do desenvolvimento de um trabalho deste tipo, a metodologia Cowboy revela-se uma alternativa válida para guiar e auxiliar na condução do

projeto. Conforme exposto na subseção anterior, o método dispõe-se a auxiliar profissionais que, devido às suas circunstâncias, necessitam programar solitariamente, o que levava muitos desenvolvedores a não adotarem metodologia alguma. Compreendendo que esse tipo de cenário de desenvolvimento solitário já era uma realidade e buscando integrar algumas práticas ao processo desses desenvolvedores, serão consideradas, neste trabalho:

- A iteratividade do processo, em que, em cada iteração, deverá constar a adição de novas características do software e com duração entre 2 semanas e 1 mês.
- A elaboração de um backlog, ou seja, um registro ou histórico de cada iteração do projeto, em que, em cada iteração, deve constar uma lista do que foi abordado nela.
- A simplicidade na elaboração dos artefatos, em que os mesmos devem ter ligação intrínseca com o negócio, incluindo documentação que, por questões de praticidade, podem ser levantadas por documentos derivados da documentação feita pelo próprio código-fonte.
- A utilização de um ambiente de desenvolvimento integrado, seguindo alguma referência de boas práticas de escrita de código, fazendo a refatoração imediatamente quando necessário e balizando o que foi produzido por testes, para averiguar o comportamento das unidades do software.

As práticas citadas acima, que estão presentes nas metodologias ágeis atualmente, estão em consonância com as necessidades do projeto na produção do protótipo deste trabalho.

6. Implementação

Esta seção irá cobrir os aspectos da implementação da rede neural profunda com o framework Tensorflow e a máquina de vetores de suporte com a biblioteca scikit-learn, ambos na linguagem ed programação Python. Para facilitar a compreensão serão utilizadas imagens do código dentro do ambiente Jupyter Notebook. O *dataset* utilizado, possui 32 colunas, sendo 30 utilizadas na classificação e a coluna 'diagnosis' do tipo string com valores 'M' para maligno e 'B' para benigno. A respeito das outras colunas e seus respectivos tipos de valores, constam nas imagens 8, 9 e 10:

radius_mean	No description provided	Numeric
texture_mean	No description provided	Numeric
perimeter_mean	No description provided	Numeric
area_mean	No description provided	Numeric
smoothness_mean	No description provided	Numeric
compactness_mean	No description provided	Numeric
concavity_mean	No description provided	Numeric
concave points_mean	No description provided	Numeric
symmetry_mean	No description provided	Numeric
fractal_dimension_mean	No description provided	Numeric
radius_se	No description provided	Numeric

Figura 8. Colunas do Dataset

texture_se	No description provided	Numeric
perimeter_se	No description provided	Numeric
area_se	No description provided	Numeric
smoothness_se	No description provided	Numeric
compactness_se	No description provided	Numeric
concavity_se	No description provided	Numeric
concave points_se	No description provided	Numeric
symmetry_se	No description provided	Numeric
fractal_dimension_se	No description provided	Numeric
radius_worst	No description provided	Numeric
texture_worst	No description provided	Numeric
perimeter_worst	No description provided	Numeric
area_worst	No description provided	Numeric

Figura 9. Colunas do Dataset

Conforme as imagens, temos apenas dados do tipo numeric, ou seja, valores não discretos. Posteriormente é feito a padronização de dados para evitar problemas como o paradoxo da acurácia ou resultados inconsistentes devido a disproporção de pesos devido aos valores absolutos de cada coluna.

smoothness_worst	No description provided	Numeric
compactness_worst	No description provided	Numeric
concavity_worst	No description provided	Numeric
concave points_worst	No description provided	Numeric
symmetry_worst	No description provided	Numeric
fractal_dimension_worst	No description provided	Numeric

Figura 10. Colunas do Dataset

6.1. Rede Neural Profunda com Tensorflow

Nesta seção trataremos da implementação em python da rede neural profunda com o framework Tensorflow.

6.1.1. Dependências do projeto e Bibliotecas

Começaremos com as importações de bibliotecas necessárias e também a leitura dos dados do *dataset* no arquivo 'data.csv':

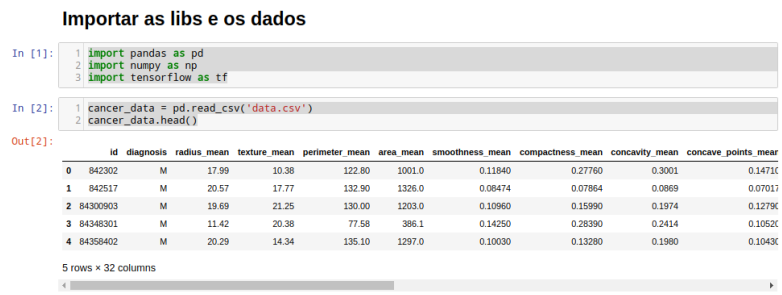

Figura 11. Imports de bibliotecas e leitura de dados

Na imagem consta a importação das bibliotecas pandas, numpy e tensorflow. Foram utilizados apelidos (alias) para cada uma delas, sendo respectivamente pd, np e tf. A biblioteca pandas é utilizada para abrir arquivos de *datasets*. Já a biblioteca numpy para calculo numérico e algebra linear e por último a bilioteca tensorflow para os algoritmos de aprendizado de máquina. Ao final da imagem é mostrado como o *dataset* está ao ser recém carregado.

6.1.2. Tratamento de dados numéricos

Após a importação precisamos começar o processo chamado de pré-processamento, onde foi feito uma série de procedimentos nos nossos dados para que eles estejam aptos para serem utilizados em um treinomento de aprendizado de máquina. Primeiro foi necessário

tratar os dados utilizando o método de padronização de dados, onde ao final do processo nossos dados terão média igual à zero e variância igual à um.

Na imagem abaixo primeiro é feito uma seleção das colunas que possuem valores numéricos, depois disso é feito nessas colunas o procedimento onde se subtrai um valor pela média daquela coluna e dividi-se o resultado disso pelo desvio padrão da coluna.

```
In [8]:
cancer_data.columns

Out[8]:
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave_points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')

In [9]:
cols_normalizar = ['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave_points_worst',
       'symmetry_worst', 'fractal_dimension_worst']

In [10]:
cancer_data[cols_normalizar] = cancer_data[cols_normalizar].apply(lambda x: (x - np.mean(x)) / (np.std(x)  ) )
```

Figura 12. Imports de bibliotecas e leitura de dados

6.1.3. Tratamento dos dados categóricos

Na imagem seguinte é feito a transformação de dados categóricos para que seja possível o trabalho dos algoritmos da biblioteca.

Limpar e normalizar os dados

```
In [4]:
1 cancer_data['diagnosis'].unique()
Out[4]: array(['M', 'B'], dtype=object)

In [5]:
1 def M_B_0_1(label):
2     if label == 'M':
3         return 1
4     else:
5         return 0

In [6]:
1 cancer_data['diagnosis'] = cancer_data['diagnosis'].apply(M_B_0_1)
```

Figura 13. Tratamento de dados Categóricos

Na primeira linha é identificado os tipos de valores categóricos presentes na tabela 'diagnosis' e depois é criada uma função para se aplicar a ela, onde os valores 'M' para maglinos são convertidos no valor numérico 1 e os valores 'B' para benignos são convertidos em 0. Na imagem seguinte é possivel ver o resultado da trasnformação no *dataset*:

```
In [7]:
1 cancer_data
Out[7]:
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave_points_me
0	842302	1	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.300100	0.1471
1	842517	1	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.086900	0.0701
2	84300903	1	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.197400	0.1279
3	84348301	1	11.420	20.38	77.58	386.1	0.14250	0.28390	0.241400	0.1052
4	84358402	1	20.290	14.34	135.10	1297.0	0.10030	0.13280	0.198000	0.1043
5	843786	1	12.450	15.70	82.57	477.1	0.12780	0.17000	0.157800	0.0808
6	844359	1	18.250	19.98	119.60	1040.0	0.09463	0.10900	0.112700	0.0740
7	84458202	1	13.710	20.83	90.20	577.9	0.11890	0.16450	0.093660	0.0598
8	844981	1	13.000	21.82	87.50	519.8	0.12730	0.19320	0.185900	0.0935
9	84501001	1	12.460	24.04	83.97	475.9	0.11860	0.23960	0.227300	0.0854
10	845636	1	16.020	23.24	102.70	797.8	0.08206	0.06669	0.032990	0.0333
11	84610002	1	15.780	17.89	103.60	781.0	0.09710	0.12920	0.099540	0.0666
12	846226	1	19.170	24.80	132.40	1123.0	0.09740	0.24580	0.206500	0.1118
13	846381	1	15.850	23.95	103.70	782.7	0.08401	0.10020	0.099380	0.0536
14	84667401	1	13.730	22.61	93.60	578.3	0.11310	0.22930	0.212800	0.0803
15	84799002	1	14.540	27.54	96.73	658.8	0.11390	0.15950	0.163900	0.0736

Figura 14. Após o tratamento de dados categóricos

6.1.4. Tratamento dos dados no Tensorflow

Após o tratamento da coluna 'dianogis' é visto quais os nomes de cada coluna e para uma delas é criada uma variável reconhecível pelo tensorflow, lembrando que precisam ser do mesmo tipo, por isso é utilizada o tipo $tf.feature_column.numeric_column$ que é equivalente para tipos numéricos no Tensorflow. Na figura abaixo, consta essa transformação e a retirada da coluna ID pois ela não possui relevância ou correlação com os dados amostrais, tratando-se apenas de identificador. Também é feita a divisão dos dados entre valores de entrada e resultados, sendo cada um guardados nas variáveis `x_data` e `y_data`.

6.1.5. Separação dos datasets de treino e teste

Em seguida são feitas as divisões de *datasets* entre treino e testes, para que o modelo treinado seja testado com valores que o mesmo desconhece e assim tenhamos métricas

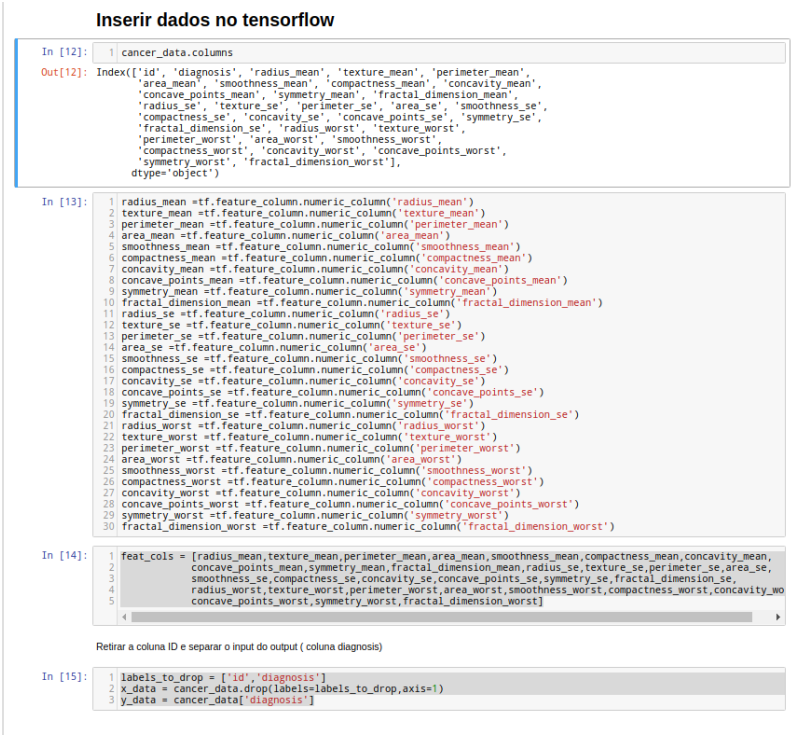

Figura 15. Inserção das colunas no tensorflow

mais confiáveis nos resultados.

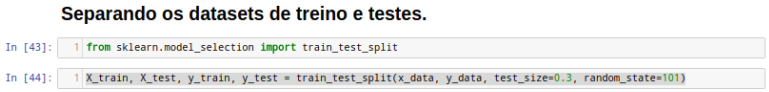

Figura 16. Separação de dataset de treino e de testes

Conforme está na imagem, é feito uma importação do método `train_test_split` da biblioteca scikit-learn para efetuar a separação. No método colocaremos 70% do *dataset* original para treinamentos e 30% para testes.

6.1.6. Criação do modelo de Rede Neural e vinculação com os dados de treino

Em sequência é criado a função para receber os dados dos valores criados no tensorflow, a qual iremos utilizar para treinar o modelo criado de rede neural profunda.

Na variavel `input_func`, é atribuido o valor da nossa equação de treino do tensorflow, misturando a ordem os dados e colocando em grupos de 10 amostras por lote (batches). A variavel model receberá o nosso modelo de rede neural profunda, que corresponde a classe `DNNClassifier` do Tensorflow. A rede neural possuirá 3 camadas intermediárias de 200 neurônios. No último comando é feito o treinamento de 1000 vezes do modelo pelo *dataset*, atribuindo a função que colocamos no início.

Treinando com o tensorflow

```
In [45]: input_func = tf.estimator.inputs.pandas_input_fn(x=X_train,y=y_train,batch_size=10,num_epochs=1000,shuffle=True)

In [46]: model = tf.estimator.DNNClassifier(hidden_units=[200,200,200],feature_columns=feat_cols,n_classes=2)

INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmpz977db3c
INFO:tensorflow:Using config: {'_save_checkpoints_steps': None, '_keep_checkpoint_every_n_hours': 10000, '_keep_chec
kpoint_max': 5, '_log_step_count_steps': 100, '_session_config': None, '_model_dir': '/tmp/tmpz977db3c', '_save_summ
ary_steps': 100, '_save_checkpoints_secs': 600, '_tf_random_seed': 1}

In [47]: model.train(input_fn=input_func,steps=1000)

INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Saving checkpoints for 1 into /tmp/tmpz977db3c/model.ckpt.
INFO:tensorflow:step = 1, loss = 6.94218
INFO:tensorflow:global_step/sec: 101.489
INFO:tensorflow:step = 101, loss = 0.0302808 (1.006 sec)
INFO:tensorflow:global_step/sec: 126.025
INFO:tensorflow:step = 201, loss = 0.000413091 (0.779 sec)
INFO:tensorflow:global_step/sec: 132.517
INFO:tensorflow:step = 301, loss = 0.0446868 (0.765 sec)
INFO:tensorflow:global_step/sec: 142.297
INFO:tensorflow:step = 401, loss = 0.000189935 (0.698 sec)
INFO:tensorflow:global_step/sec: 128.206
INFO:tensorflow:step = 501, loss = 0.00021193 (0.784 sec)
INFO:tensorflow:global_step/sec: 117.411
INFO:tensorflow:step = 601, loss = 0.000603641 (0.838 sec)
INFO:tensorflow:global_step/sec: 133.809
INFO:tensorflow:step = 701, loss = 1.3477e-07 (0.754 sec)
INFO:tensorflow:global_step/sec: 134.664
INFO:tensorflow:step = 801, loss = 0.000329931 (0.750 sec)
INFO:tensorflow:global_step/sec: 120.326
INFO:tensorflow:step = 901, loss = 7.24648e-07 (0.820 sec)
INFO:tensorflow:Saving checkpoints for 1000 into /tmp/tmpz977db3c/model.ckpt.
INFO:tensorflow:Loss for final step: 1.29416e-07.

Out[47]: <tensorflow.python.estimator.canned.dnn.DNNClassifier at 0x7fb453664f98>
```

Figura 17. Separação de dataset de treino e de testes

6.1.7. Avaliando a Rede Neural Profunda

Para o treinamento foi utilizado os *datasets* de treino `X_train` e `y_train`. Para os testes e a avaliação ou `evaluation` vamos utilizar os *datasets* `X_test` e `y_test`. Na imagem a seguir temos o exemplo da função de avaliação sendo criada e dos resultados obtidos pelo teste:

```
In [48]: eval_input_func = tf.estimator.inputs.pandas_input_fn(x=X_test,
                                                              y=y_test,
                                                              batch_size=10,
                                                              num_epochs=1,
                                                              shuffle=False)

In [49]: results = model.evaluate(eval_input_func)

WARNING:tensorflow:Casting <dtype: 'float32'> labels to bool.
WARNING:tensorflow:Casting <dtype: 'float32'> labels to bool.
INFO:tensorflow:Starting evaluation at 2017-10-29-19:57:48
INFO:tensorflow:Restoring parameters from /tmp/tmpz977db3c/model.ckpt-1000
INFO:tensorflow:Finished evaluation at 2017-10-29-19:57:49
INFO:tensorflow:Saving dict for global step 1000: accuracy = 0.964912, accuracy_baseline = 0.614035, auc = 0.974531,
auc_precision_recall = 0.981984, average_loss = 0.470191, global_step = 1000, label/mean = 0.385965, loss = 4.46681,
prediction/mean = 0.374496

In [50]: results

Out[50]: {'accuracy': 0.9649123,
 'accuracy_baseline': 0.61403513,
 'auc': 0.97453105,
 'auc_precision_recall': 0.98198426,
 'average_loss': 0.47019103,
 'global_step': 1000,
 'label/mean': 0.3859649,
 'loss': 4.4668145,
 'prediction/mean': 0.37449628}
```

Figura 18. Avaliação do Modelo

Os resultados apontam para 96% de acurácia e um AUC de 0.97, lembrando que 1.0 seria o valor de um classificador perfeito, ou seja, nosso classificador com base em rede neural alcançou altos valores de eficácia para esse *dataset*.

6.2. Implementação da SVM

Nesta seção iremos tratar da implementação da Máquina de suporte de vetores (SVM) e os tratamentos de dados relacionados a essa implementação.

6.2.1. Importação de bibliotecas e dependências

Para desenvolver a implementação da máquina de suporte de vetores, foram utilizadas também as bibliotecas pandas para abertura de arquivos com os dados disponibilizados, numpy para algebra e cálculos diversos e algumas funções e classes da scikit-learn para divisão de *datasets* e para instanciar a SVM.

Imports

In [27]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import matplotlib.lines as mlines
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
%matplotlib inline
```

Figura 19. Bibliotecas importadas do sistema

6.2.2. Leitura dos dados e Tratamento de dados categóricos

De forma semelhante a rede neural profunda, o primeiro passo para iniciarmos o algoritmo é ler os dados no nosso arquivo `data.csv` e começar o tratamento de pré-processamento. Para isso foi usada a função `read_csv` da biblioteca pandas para ler os dados do arquivo.

Transformando tumor "B" (benigno) em 0 e "M" (maligno) em 1

In [28]:
```

#encoding the y-label
def M_B_0_1(label):
    if label == 'M':
        return 1
    else:
        return 0

```

In [29]:
```
cancer_data = pd.read_csv('data.csv')
cancer_data['diagnosis'] = cancer_data['diagnosis'].apply(M_B_0_1)
```

Figura 20. Tratamento de dados categóricos

Em seguida foi utilizada a função `M_B_0_1` para transformar os dados da coluna 'diagnosis' que antes possuia valores 'M' para tumores malignos e 'B' para tumores benignos, para os valores 1 e 0 respectivamente.

6.2.3. Tratamento de dados numéricos

No tratamento de dados numéricos, foi aplicado também a padronização de dados. O procedimento funciona da mesma forma da rede neural profunda, sendo assim ambas

implementações trabalham sobre uma mesma base de dados normalizados. Lembrando que a padronização faz com que os valores das colunas numéricas tenham como propriedades, média igual à zero e variância igual à um. O código está na figura abaixo:

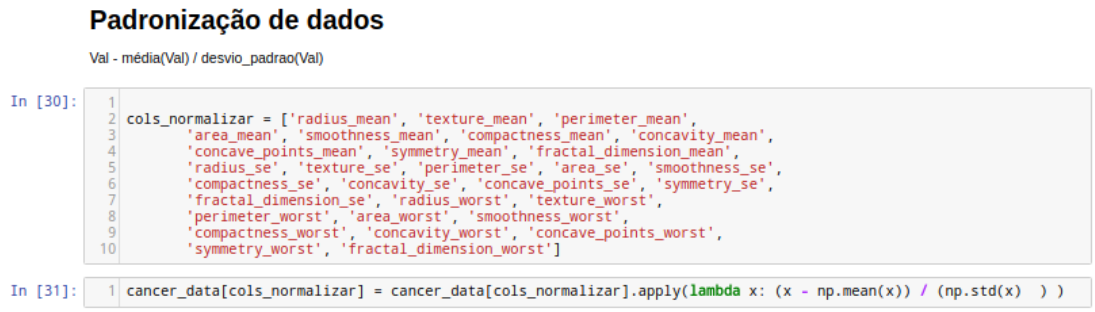

Figura 21. Tratamento de dados Numéricos

6.2.4. Separando colunas dos dados e resultados

Conforme foi feito também na rede neural profunda, os dados das propriedades dos tumores foram alocados na variável `x_data` e os seus respectivos diagnósticos na variável `y_data`. Também foi retirada a coluna 'id' pois ela não influencia na correlação dos tumores e seus diagnósticos.

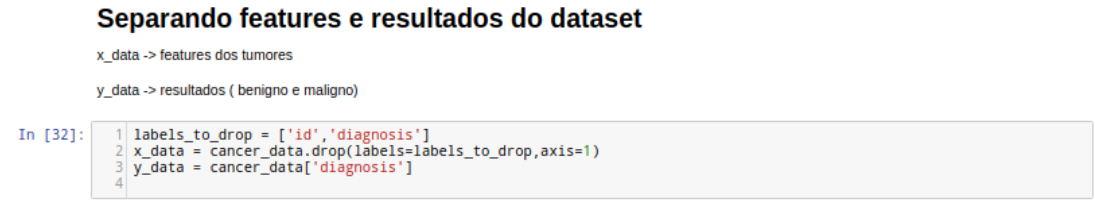

Figura 22. Separando propriedades e diagnósticos do dataset

6.2.5. Separação de Datasets de Treino e Testes

De forma semelhante foi utilizado também a função `train_test_split` da biblioteca scikit-learn para separar o *dataset* original em dois *datasets*, um para treino do modelo e outro para testes.

Dessa separação criam-se dois *datasets*, o de treino representado pelas variáveis `X_train` e `y_train` e o *dataset* de testes representado por `X_test` e `y_test`.

6.2.6. Criando Classificador SVM

No próximo passo instanciamos a classe SVC com o kernel 'rbf' que é um dos parâmetros de criação da SVM. Em seguida invocamos a função `fit` para treinar nosso modelo de SVM com os dados de treino.

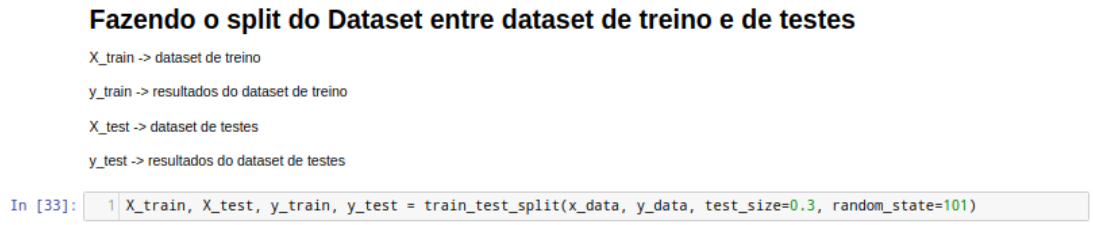

Figura 23. Separação do dataset de treino e testes

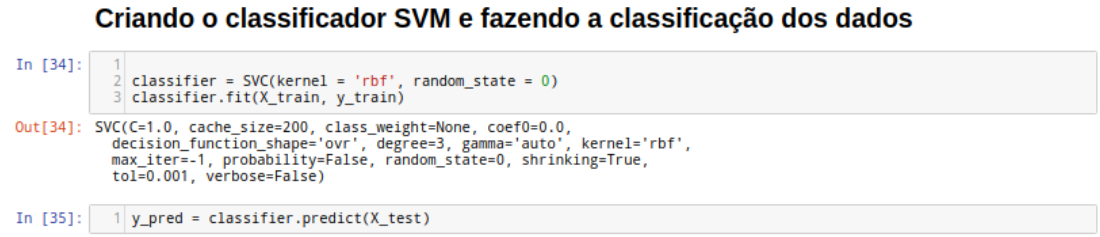

Figura 24. Criando classificador e fazendo teste de predição

Ao final é feito uma predição com os dados de teste em `X_test` dentro da função `predict` da classe `SVC`, armazenando os resultados da predição na variável `y_pred`.

6.2.7. Imprimindo valores dos resultados

Na impressão dos valores de resultado, foi utilizada a função `classification_report` e a função `accuracy_score`, ambas da biblioteca scikit-learn.

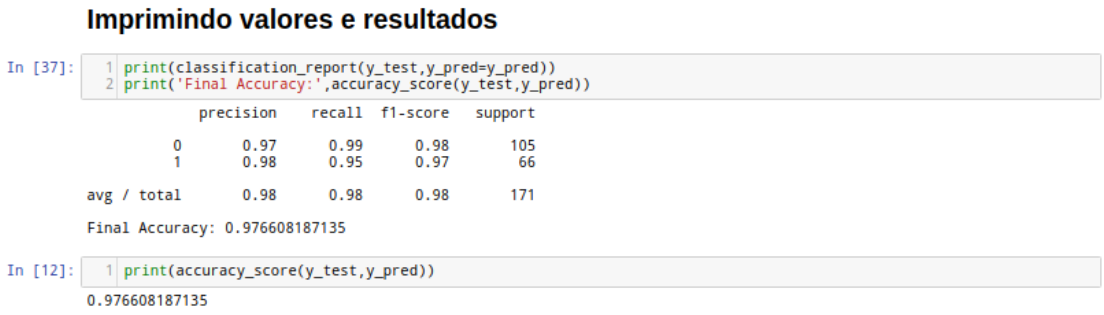

Figura 25. Resultados da SVM

O resultado final da classificação foi de 0.9766 ou 97.66% de acurácia e 0.98 de média de f1-score. Os altos índices de desempenho corroboram que esse método possui uma boa validade de uso para *datasets* com este perfil.

7. Resultados

Nesta seção iremos descrever os resultados obtidos e discutir os seus significados. Na imagem 8 temos os valores dos resultados da classificação da rede neural profunda com o grupo de dados de teste:

	precision	recall	f1-score	support
0	0.96	0.98	0.97	105
1	0.97	0.94	0.95	66
avg / total	0.96	0.96	0.96	171

Final Accuracy 0.964912280702

Figura 26. Resultado das classificações com Redes Neurais Profundas

Na imagem temos uma acurácia de 96% e um f1-score de 0.96, ou seja, mesmo com classes não perfeitamente balanceadas no *dataset*, foi possível alcançar um alto grau de desempenho. Na imagem 9 seguem os resultados da classificação da rede neural profunda utilizando o framework tensorflow:

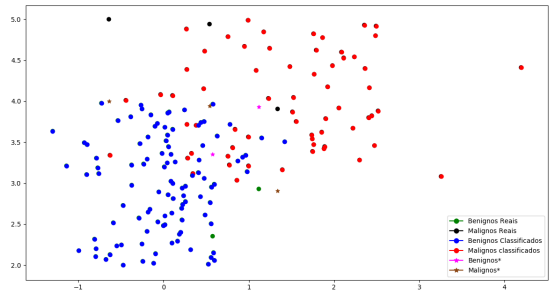

Figura 27. Gráfico das classificações com Redes Neurais Profundas

Para compreender corretamente a precisão que esse gráfico busca representar, devemos compreender que se houvessem apenas pontos vermelhos e azuis o classificador seria perfeito, ou seja 100% de acertos nos diagnósticos.

Como podemos ver existem ainda 2 pontos verdes e duas estrelas rosas, ou seja, temos 2 exemplos de tumores que eram benignos mas foram classificados como malignos. Também podemos ver 3 pontos pretos e 3 estrelas douradas, ou seja, 3 tumores malignos que foram classificados como benignos pelo algoritmo.

No caso da máquina de vetores de suporte(SVM) foram obtidos os seguintes resultados pertante a classificação do grupo de teste:

	precision	recall	f1-score	support
0	0.97	0.99	0.98	105
1	0.98	0.95	0.97	66
avg / total	0.98	0.98	0.98	171

Final Accuracy: 0.976608187135

Figura 28. Resultados de classificação da SVM

Pelo resultado de 97.66% de acurácia e também pelo elevado valor do indicador f1-score, é possível concluir que a SVM também teve um elevado desempenho na

classificação dos dados de teste.

Na imagem 11 está os resultados impressos conforme as classificações do grupo de teste:

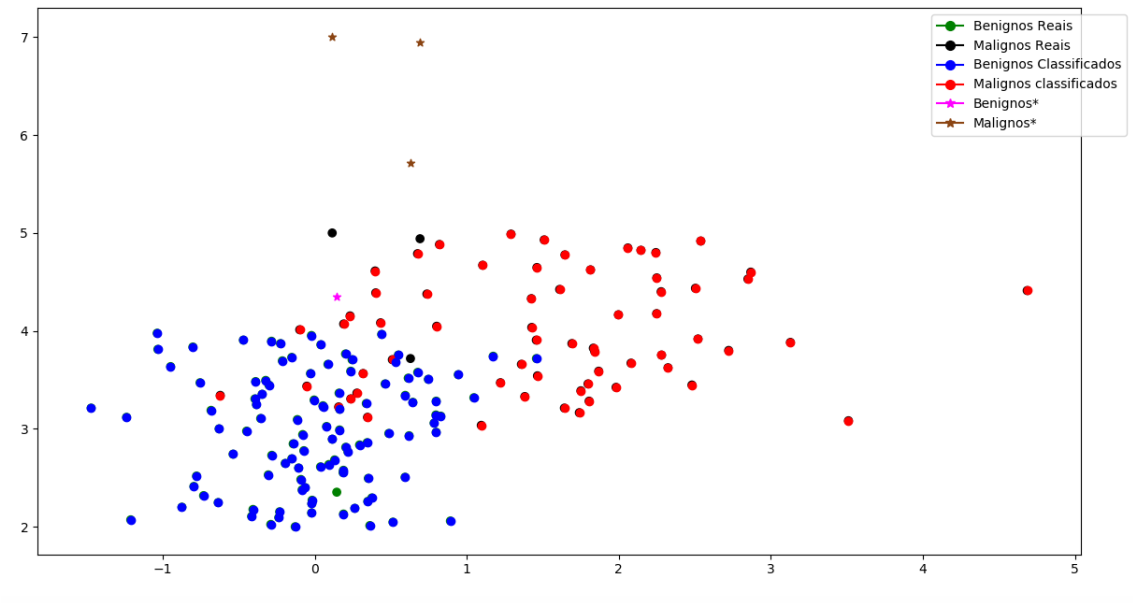

Figura 29. Gráfocp das classificações com a SVM

De forma semelhante ao gráfico das redes neurais profundas, é possível identificar um ponto verde e uma estrela rosa, o que indica que um tumor benigno foi classificado como maligno. também é possível identificar três pontos pretos e três estrelas douradas, o que significa que três tumores malignos foram classifiados como benignos.

8. Conclusões

Com resultados chegando a 96%, é visível a alta capacidade de classificação que algoritmos de aprendizado de máquina podem desempenhar em atividades como esta que é tão urgente no caso do câncer de mama. Por se tratar de um teste muito dependente do empirismo, compreende-se que as redes neurais profundas tiveram desempenho inferior ao da SVM devido ao tamanho mais enxuto dos dados analisados.

Mesmo perante a erros, que sim devem ser analisados para tentar alcançar maiores valores de desempenho dos algoritmos, é importante relembrar que trata-se de um sistema de apoio a decisão e não de tomada de decisão. É evidente que a utilização destas técnicas pelo seu alto grau de eficácia ajudam e devem ser utilizadas como um auxílio no desempenho de diagnósticos médicos do câncer de mama evitando falhas humanas como foi feito em [Wang et al. 2016].A continuação deste trabalho certamente deve ser dado um passo a mais para classificação não apenas de dados dos tumores mas também de classificação de imagens de mamografias e no auxílio de diagnóstico dessas imagens também.

Referências

Abma, B. (2009). Evaluation of requirements management tools with support for traceability-based change impact analysis. *Master's thesis, University of Twente, Enschede*.

Becker, A. S., Marcon, M., Ghafoor, S., Wurnig, M. C., Frauenfelder, T., and Boss, A. (2017). Deep learning in mammography: diagnostic accuracy of a multipurpose image analysis software in the detection of breast cancer. *Investigative Radiology*, 52(7):434–440.

Blum, A. (2007). Machine learning theory. *Carnegie Melon Universit, School of Computer Science*, page 26.

da Silva, I. N. D. C. J. A. G. INCA-2017.

Fawcett, T. (2004). Roc graphs: Notes and practical considerations for researchers. *Machine learning*, 31(1):1–38.

Hollar, A. B. (2006). Cowboy: An agile programming methodology for a solo programmer.

Melo, M., Gajadhar, A., and Batista, L. V. (2014). Análise comparativa de métodos de aprendizagem de máquina para classificaçao de massas em mamografias. In *Congresso da Sociedade Brasileira de Computaçao-Workshop de Informática Médica*, pages 1772–1775.

Sommerville, I. (2010). *Software engineering*. Pearson.

Summerfield, M. (2010). *Programming in Python 3: a complete introduction to the Python language*. Addison-Wesley Professional.

Wang, D., Khosla, A., Gargeya, R., Irshad, H., and Beck, A. H. (2016). Deep learning for identifying metastatic breast cancer. *arXiv preprint arXiv:1606.05718*.