Denner Araujo Costa

Auxílio ao Diagnóstico da Diabetes Tipo II na Região Nordeste com a Utilização de Marcadores Não Invasivos.

São Luís - MA

2017

Denner Araujo Costa

Auxílio ao Diagnóstico da Diabetes Tipo II na Região Nordeste com a Utilização de Marcadores Não Invasivos.

Trabalho apresentado ao Programa de Pós-Graduação em Engenharia de Computação e Sistemas do Departamento de Engenharia de Computação da Universidade Estadual do Maranhão como requisito parcial para obtenção do grau de Mestre em Engenharia de Computação.

Universidade Estadual do Maranhão – UEMA

Centro de Ciências Tecnológicas

Pós-Graduação em Engenharia de Computação e Sistemas

Orientadora: Profa. Dra. Áurea Celeste da Costa Ribeiro

São Luís - MA

2017

Costa, Denner Araujo
Auxílio ao Diagnóstico da Diabetes Tipo II na Região Nordeste com a Utilização de Marcadores Não Invasivos./ Denner Araujo Costa. – São Luís, 2017.
81 p.

Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia de Computação e Sistemas, Universidade Estadual do Maranhão, 2017.

Orientadora: Profa. Dra. Áurea Celeste da Costa Ribeiro

1. *Data Mining*. 2. Diabetes. 3. SUS. 4. Brasil. I. Título

CDU 004.6:616.379-008.64(81)

Denner Araujo Costa

Auxílio ao Diagnóstico da Diabetes Tipo II na Região Nordeste com a Utilização de Marcadores Não Invasivos.

Trabalho apresentado ao Programa de Pós-Graduação em Engenharia de Computação e Sistemas do Departamento de Engenharia de Computação da Universidade Estadual do Maranhão como requisito parcial para obtenção do grau de Mestre em Engenharia de Computação.

Trabalho defendido. São Luís - MA, 8 de Agosto de 2017:

Profa. Dra. Áurea Celeste da Costa Ribeiro
Orientadora

Examinador 1
Dr. Cícero Costa Quarto

Examinador 2
Dr. Fernando Jorge Cutrim Demetrio

São Luís - MA
2017

A todos que direta ou indiretamente fizeram parte de minha formação.

AGRADECIMENTOS

A minha orientadora Profa. Dra. Áurea Celeste da Costa Ribeiro pelo suporte, pela suas correções e incentivos.

A Giselle pelo suporte e críticas realizadas ao trabalho sem as quais a tinta utilizada no mesmo seria 40% menor.

A todos que direta ou indiretamente fizeram parte de minha formação meu muito obrigado.

É perigoso ir sozinho.
(The Legend of Zelda (NES))

RESUMO

Na atualidade, a Diabetes Mellitus é uma patologia incurável e o número de pessoas afetadas continua a agravar-se. Neste trabalho foram aplicadas técnicas de *data mining* e codificação eficiente para extrair conhecimento de uma amostra dos dados existentes no *Data Warehouse* do Sistema Único de Saúde para diagnosticar diabetes sem a utilização de valores de testes bioquímicos. A codificação eficiente foi utilizada para formar uma nova representação concisa dos dados e diferentes técnicas de *data mining* foram utilizadas para demonstrar a melhora na classificação das classes. O método proposto alcançou acurácia de 99.96%.

Palavras-chave: *Data Mining*. Diabetes. SUS. Brasil.

ABSTRACT

Currently, Diabetes Mellitus is an incurable disease and the number of affected people continues to worsen. In this work data mining and efficient coding techniques were applied to extract knowledge from a sample of the data in the Single Health System Data Warehouse to diagnose diabetes without the use of biochemical test values. Efficient coding was used to form a new concise representation of the data and different data mining techniques were used to demonstrate the improvement in class classification. The proposed method reached an accuracy of 99.96%.

Keywords: Data Mining. Diabetes. SHS. Brazil.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

DM	Diabetes Mellitus
DM1	Diabetes Mellitus tipo 1
DM2	Diabetes Mellitus tipo 2
DMG	Diabetes Mellitus Gestacional
ICA	*Independent Component Analysis*
IFG	*Impaired Fasting Glucose*
IGT	*Impaired Glucose Tolerance*
KNN	*k-Nearest Neighbours*
NHANES	Inquérito de Saúde e Nutrição Exame Nacional do inglês *National Health and Nutrition Examination Survey*
PRAHADM	Plano de Reorganização da Atenção à Hipertensão Arterial e Diabetes Mellitus
SisHperdia	Sistema de Cadastramento e Acompanhamento de Hipertensos e Diabéticos
SUS	Sistema Único de Saúde
SVM	*Support Vector Machine*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo começa com a descrição do problema e a motivação para realizar esta pesquisa. Em seguida, é apresentada a principal questão de pesquisa. Encerrando esse capítulo com as principais contribuições de pesquisa.

1.1 Descrição do problema

Esta dissertação visa estudar os dados sobre a doença Diabetes Mellitus, aplicando métodos de Tecnologias da Informação e do Conhecimento, integrados na área científica de *Business Intelligence* da qual fazem parte, e as subáreas *Data Warehousing* e *Data Mining*.

As organizações atuais têm hoje ao seu dispor tecnologias de informação que lhes permite conhecer melhor e gerir bem os seus processos de negócio (BORGES, 2016). A contínua redução nos custos de armazenamento e processamento de informação, permitiu que as organizações possuam hoje uma enorme quantidade de dados sobre as suas atividades.

As aplicações de mineração de dados nos serviços medicinais, auxilia na redução de erros clínicos, identificação precoce, ação contra-ativa ao combate de doenças e redução de custos futuros pelo agravamento da doença (YASASWI; PRAJNA, 2016).

Este estudo concentra-se na doença Diabetes Mellitus (DM), por ser uma doença ainda incurável e pelo fato que o número de pessoas afetadas continua a aumentar mundialmente como o comprova a (IDF - Federation Internation Diabetes, 2015) e no Brasil (MILECH et al., 2016). Esta ideia é corroborada pelo estudo de (BROWN et al., 2012) e (RIBEIRO et al., 2015a) especialmente em casos não diagnosticados para intervenção o mais cedo possível. Os autores referem o interesse em encontrar sistemas de identificação de pacientes sem recorrer a testes bioquímicos. O desenvolvimento de um sistema como este pode fazer decrescer o número de pacientes não diagnosticados.

A DM inclui um grupo de doenças metabólicas caracterizado por hiperglicemia, resultante da disfunção na secreção de insulina e/ou em sua ação (III et al., 1997). A hiperglicemia crônica é o principal fator desencadeador das complicações da DM. É comum o desenvolvimento das macroangiopatias, que comprometem as artérias coronarianas, dos membros inferiores e as cerebrais. Outras complicações também são conhecidas na DM e englobam as microangiopatias, afetando, especificamente, a retina, o glomérulo renal e os nervos periféricos (CANANI et al., 2004) (BARBOSA; OLIVEIRA; SEARA, 2009) (FERREIRA et al., 2011) (American Diabetes Association et al., 2014) .

Vários processos patogênicos estão envolvidos no desenvolvimento da diabetes. Estes

vão desde a destruição crônica das células β pancreáticas, com a consequente deficiência de insulina até anormalidades que resultam em resistência à ação da insulina (FERREIRA et al., 2011). A base das anormalidades no metabolismo de carboidratos, gorduras e proteínas no diabetes é a ação deficiente da insulina nos tecidos alvo.

A insuficiência de ação da insulina, resulta da secreção inadequada de insulina, diabetes tipo 1(DM1), ou da diminuição das reações teciduais à insulina, diabetes tipo 2(DM2), em um ou mais pontos nas complexas vias de ação hormonal (American Diabetes Association et al., 2014). A deficiência de secreção de insulina e defeitos na ação da insulina coexistem frequentemente no mesmo paciente, e nem sempre é claro qual anormalidade, se sozinha, é a principal causa da hiperglicemia.

De acordo com American Diabetes Association et al. (2014) os sintomas de hiperglicemia acentuada incluem poliúria[1], polidipsia[2], perda de peso, por vezes com polifagia[3], e visão turva. Prejuízo ao crescimento e susceptibilidade a certas infecções também podem acompanhar hiperglicemia crônica. As consequências agudas e o risco de vida do paciente com diabetes não controlado são a hiperglicemia com cetoacidose ou a síndrome hiperosmolar não cetótica.

Ainda de acordo com American Diabetes Association et al. (2014) as complicações de longo prazo do diabetes incluem retinopatia com potencial perda de visão; Nefropatia que leva à insuficiência renal; Neuropatia periférica com risco de úlceras do pé, amputações e junta de Charcot além da neuropatia autonômica que causa sintomas gastrointestinais, geniturinários, cardiovasculares e disfunção sexual. Os pacientes com diabetes apresentam um aumento da incidência de doença cardiovascular aterosclerótica, arterial periférica e doença vascular cerebral. Hipertensão e anormalidades do metabolismo das lipoproteínas são frequentemente encontradas em pessoas com diabetes.

A grande maioria dos casos de diabetes caem em duas grandes categorias etiopatogênicas. A primeira, a DM1, é uma deficiência absoluta da secreção de insulina (American Diabetes Association et al., 2014). Os indivíduos com maior risco de desenvolver este tipo de diabetes podem frequentemente ser identificados pela evidência sorológica de um processo patológico auto-imune que ocorre nos ilhéus pancreáticos e de marcadores genéticos.

A segunda categoria mais prevalente, a DM2, é uma combinação de resistência à ação da insulina e uma resposta secretora de insulina compensatório inadequado. Esta última categoria apresenta um valor de hiperglicemia suficiente para causar alterações patológicas e funcionais em vários tecidos-alvo, mas sem sintomas clínicos, pode estar presente durante um longo período de tempo antes que a DM2 seja detectada (American

[1] Poliúria é um sintoma em que há produção de urina acima de 2,5 litros por dia.
[2] Polidipsia é um sintoma em que há uma excessiva sensação de sede.
[3] Polifagia ou hiperfagia é fome excessiva ou aumento do apetite

Diabetes Association et al., 2014). Durante este período assintomático é possível verificar uma anormalidade no metabolismo de carboidratos por medição da glicemia plasmática em jejum ou após um teste oral de tolerância à glicose.

1.1.1 Custo da Diabetes

Segundo Barcelo et al. (2003) dado sua natureza crônica, que pode levar a complicações graves, o DM2 é uma condição onerosa tanto para os pacientes quanto para o sistema de saúde. Bahia et al. (2011) avaliou o custo do DM2 entre pacientes ambulatoriais brasileiros e demonstrou que os custos aumentaram conforme a progressão do diabetes e a presença de complicações crônicas.

Saraiva et al. (2016) realizou um estudo para descrever a utilização de recursos de saúde associados ao tratamento do DM2 no sistema público de saúde brasileiro. Neste estudo foi observado um custo total médio maior entre pacientes com complicações (R$ 1.212,37) quando comparados àqueles sem complicações (R$ 931,88), esses valores se apresentam em concordância com o achado dos estudos de Bahia et al. (2011) e Marinho et al. (2011), previamente descrito para a população brasileira com DM2, no qual um custo incremental de 25% naqueles com complicações tanto microvasculares quanto macrovasculares foi reportado.

Os custos indiretos com a doença não foram orçados nestes estudo. No Brasil, há uma carência de estudos para que se possa medir os custos reais do diabetes, bem como sua incidência de mortalidade e prevalência. Os recursos disponíveis no sistema de saúde são limitados, desta forma, torna-se essencial a correta administração destes para um melhor aproveitamento e para que possam ser planejadas ações de combate a doença (ABREU et al., 2008).

1.2 Questão de Pesquisa e Maiores Contribuições

À luz dos motivos descritos na seção anterior, a principal questão de pesquisa abordada nesta dissertação é a seguinte:

É possível identificar diabetes em indivíduos etnicamente heterogêneos, mesmo sem recorrer a testes bioquímicos?

Embora existam trabalhos sobre a tentativa de identificação de diabetes sem testes bioquímicos, para o melhor conhecimento do autor, uma tentativa de diagnosticar diabetes em uma região tão extensa e etnicamente heterogenia como é o caso dos estados da região nordeste do Brasil nunca foi feito antes.

Esta dissertação, portanto, apresenta um trabalho original com o seguinte como suas principais contribuições:

- **Abordagem para lidar com o problema descrito:** Um dos trabalhos anteriores realizou a tentativa de predição de diabetes em paciente do estado do Maranhão. A abordagem utilizada demonstrou o a viabilidade da abordagem para a população de um único estado. Esta pesquisa, por outro lado, foca na predição da diabetes para todos os nove estados da região nordeste do Brasil.

- **Abordagem para resolver o problema descrito:** O Sistema Único de Saúde (SUS), tem dados de todos os pacientes atendidos em sua rede e por programas próprios ou governamentais.
  Minha abordagem para resolver este problema envolve, com a cooperação do SUS, em utilizar a técnica de codificação eficiente para formar uma representação concisa da base de dados fornecida pelo SUS e posterior comparação de classificadores comumente utilizados em bases de saúde, comparar seus resultados e avaliar a viabilidade da predição sem testes bioquímicos, em uma população heterogenia

- **Diabetes, se conhecido no estágio inicial, pode ajudar a tomar precauções prévias e manter o nível de açúcar no sangue controlado:** Uma solução para o problema descrito é importante, pois iria fornecer ao paciente mais tempo, se detectada cedo, para controlar a diabetes e evitar que ele se agrave.

1.3 Objetivo

Desenvolver um método não invasivo de rastreamento para diabetes tipo 2 na região Nordeste do Brasil, por meio de marcadores não invasivos da doença com confiabilidade e baixo custo.

1.3.1 Objetivos Especifico

- Revisar o estado da arte;
- Selecionar base de dados;
- Identificar as melhores técnicas comumente utilizadas;
- Rastrear de forma não invasiva o diabetes tipo 2 na base selecionada;
- Verificar a viabilidade da utilização da técnica.

1.4 Organização do texto

Salvo esse capítulo, o restante dessa dissertação está organizado em cinco capítulos.

O Capítulo 2 descreve a base de dados utilizada, a técnica e algoritmo utilizado para a minimização estatística por codificação eficiente. Em seguida, os classificadores comumente utilizados em bases de saúde e utilizados nessa pesquisa são descritos. Encerrando esse capítulo com a técnica de pré-processamento utilizada.

O Capítulo 3 descreve a metodologia utilizada no trabalho, onde será detalhada as fases de seleção de dados, pré-processamento e limpeza, transformação dos dados, classificação, e a forma de avaliação dos resultados obtidos.

Os resultados obtidos são apresentados no Capítulo 4 e suas discussões no Capítulo 5.

Finalizando, o Capítulo 6 conclui a dissertação sumarizando os cinco capítulos e os resultados obtidos. Esse capitulo encerra-se apresentando as propostas de trabalhos futuro.

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo é apresentado a base de dados escolhida para essa pesquisa, além das técnicas de pré-processamento, que foi aplicada a base, de classificação, que serão testadas, e o estado da arte sobre o auxilio do diagnóstico da diabetes auxiliado por computador.

2.1 Base de Dados

As ações para o fortalecimento do sistema de vigilância em saúde estão entre as diretrizes e recomendações do Ministério da Saúde para o cuidado integral das doenças crônicas não transmissíveis (CORREIA; PADILHA; VASCONCELOS, 2014). Desse modo, faz-se necessário agregar informações sobre a situação de saúde da população, seja com relação a aspectos da mortalidade ou fatores de risco, visando definir vulnerabilidades e promover melhor ajuste das estratégias de informação (MALTA; MERHY, 2010).

O Sistema de Cadastramento e Acompanhamento de Hipertensos e Diabéticos (SisHiperdia) foi o sistema de monitoramento de programas instituído pelo Ministério da Saúde. Foi implantado em 2002, como parte do Plano de Reorganização da Atenção à Hipertensão Arterial e Diabetes Mellitus (PRAHADM), é uma plataforma que permite cadastrar e acompanhar portadores de hipertensão arterial e/ou diabetes mellitus, captados e vinculados às unidades de saúde ou equipes da atenção básica do Sistema Único de Saúde (SUS), gerando informações para profissionais e gestores de secretarias municipais e estaduais e Ministério da Saúde (INSTITUCIONAIS, 2001).

A Tabela 1 apresenta alguns dos campos encontrados nessa base. O primeiro item contém a unidade federativa, os itens 2, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16 e 17 são marcadores não invasivos. Os marcadores invasivos são 9 e 19.

Os marcadores da Tabela 1 podem ser classificados em dois tipos em relação ao tipo de dados. Os marcadores números são 2, 4, 5, 6, 7, 8, 9, com valores no conjunto de números naturais. Os textuais são 1, 3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19. Com exceção do marcador 1, que contém a informação da unidade federativa do paciente, todos os marcadores textuais são binariados, o marcador 3, contém a informação do sexo do paciente, com o sexo feminino sendo representado por (F) e o masculino por (M), os demais marcadores textuais tem o valor sim (S) ou não (N).

O Grupo dos pacientes utilizados nesse trabalho fazem parte da região nordeste do Brasil. Esta região possui um território de 1.556.001 km^2 (18,2% do território nacional). No qual tem o terceiro maior PIB do Brasil entre as grandes regiões. Sua população é superior a 50 milhões de habitantes.

Tabela 1 – Macadores da base de dados utilizada.

Número	**Marcador**	**Invasivo**	**Não invasivo**
1	Unidade Federativa		X
2	Idade (anos)		X
3	Sexo		X
4	PAS (mmHg)		X
5	PAD (mmHg)		X
6	Cintura (cm)		X
7	Peso (Kg)		X
8	Altura (cm)		X
9	Glicemia em jejum (mg/dl)	X	
10	Antecedentes familiares		X
11	Tabagismo		X
12	Sedentarismo		X
13	Sobrepeso		X
14	Infarto		X
15	Outras coronariopatias		X
16	AVC		X
17	Pé diabetico		X
18	Amputação		X
19	Doença Renal	X	

Fonte: Produzido pelos autores.

A região possui nove estados: Alagoas, Bahia, Ceará, Maranhão, Paraíba, Pernambuco, Piauí, Rio Grande do Norte e Sergipe.

2.2 Pré-Processamento

Os repositórios em saúde possuem dados que são muitas vezes confusos, com distribuições anormais, com dados em falta e variáveis grosseiramente definidas (CASTELLANI; CASTELLANI, 2003). Isto faz com que a preparação dos dados para processamento seja uma fase de vital importância e muitas vezes a razão determinante do sucesso. De acordo com a tarefa pretendida, existem algumas orientações que podem ser seguidas para minimizar a questão do ruído nos dados.

Na tarefa de classificação, o objetivo é descobrir quais atributos e seus valores que são determinantes na definição da classe. Os conjuntos de dados podem conter atributos que não servem a este objetivo, neste caso estes devem ser tratados como ruído nos dados, que pioram a performance dos modelos. Alguns exemplos de ruídos são, dados redundantes

como a idade e a data de nascimento ou irrelevantes como o gênero sexual num modelo de classificação de câncer de próstata, estes devem ser corrigidos antes da aplicação do algoritmo. Existem técnicas como a análise de correlação ou detecção dos atributos mais relevantes que permitem lidar com dados errôneos, porém estas técnicas têm limites a partir dos quais é comprometida a qualidade dos resultados (CASTELLANI; CASTELLANI, 2003). Estas técnicas analisam atributos de forma individual e embora um atributo só por si possa ser considerado ruído, a sua combinação com outros pode ter muita relevância (BORGES, 2016). É necessário ter cuidado com o uso de técnicas que eliminam atributos considerados ruídos.

Realizar uma tarefa de classificação de forma exaustiva poderia significar centenas de modelos gerados. No teste realizado por (BORGES; MARQUES; BERNARDINO, 2013) é possível observar que, quando sujeitos aos mesmos algoritmos e ferramentas, diferentes conjuntos de dados podem determinar diferentes melhores algoritmos classificadores. Na quase impossibilidade de se realizarem todos os testes para um conjunto de dados, é sempre possível existir um algoritmo que melhore a classificação. Não é praticável testar todos os algoritmos, mas é recomendável testar aqueles com os quais é normalmente possível obter, se não o melhor resultado, pelo menos um valor muito aproximado.

Sempre que os conjuntos de dados originais são de elevada dimensão é comum serem usadas técnicas estatísticas de amostragem. Sobre a amostra (conjunto de dados) é ainda realizado um particionamento de dados que divide a amostra para o treino e teste do modelo. É possível que os casos do conjunto de treino sejam de classificação muito fácil ou muito difícil (YOO et al., 2012). Por esse motivo e uma vez que não é aceitável testar o modelo com os casos usados no seu treino, deve ser usada uma técnica de particionamento como as seguintes:

- *k-fold cross-validation* – Esta técnica consiste em particionar o conjunto de dados em k-subconjuntos, normalmente 5 ou 10. São criados k-modelos e em cada um são usados k-1 subconjuntos para treino deixando de fora um dos subconjuntos para teste, por isso também chamada de "*leave-one-out*". Cada modelo gerado é testado com um subconjunto diferente. A média dos resultados dos k-modelos é habitualmente considerada o resultado do modelo final;
- *Holdout* – Neste modo de particionamento é comum falar-se em percentagens. Um *percentage split* de 70:30 significa que 70% dos casos são aleatoriamente selecionados para integrar o conjunto de dados de treino e os restantes 30% para teste.

O *k-fold cross-validation* é mais usado em conjuntos de dados pequenos e permite que os casos sejam usados tanto para treinar o modelo como para o testar. É importante lembrar que o objetivo é classificar casos novos não representados no conjunto de dados.

Relativamente a tarefas de *clustering*, quando o conhecimento sobre os dados é pouco ou nenhum. Os *outliers* podem ser interessantes em diversos domínios de aplicação, mas em *clustering* podem degradar as soluções e devem ser eliminados, caso se trate de ruído, ou considerados relevantes, caso um especialista do domínio assim o entenda.

Alguns algoritmos apenas tratam dados numéricos, o que causa a necessidade de conversão dos dados categóricos. Esta conversão pode, no entanto, resultar em distorções quando avaliada a sua distância. Tomemos como exemplo os diagnósticos para os três tipos de Diabetes: Diabetes gestacional, DM1 e DM2. Quando convertidos para um valor numérico resultaria em 1, 2 e 3, respetivamente. Usando uma medida de distância unidimensional, significaria que a relação entre a Diabetes Gestacional e a DM1 era menor (|1-2|=1) que a Diabetes Gestacional e a DM2 (|1-3|=2). É essencial que a conversão respeite os níveis de correlação existentes nos dados. Considerando a regra de associação "quem compra pão e manteiga habitualmente compra leite", verificamos que na realidade estes produtos não existem nas prateleiras do supermercado visto que são categorias de produtos e não produtos em si (BORGES, 2016). A geração de regras com todos os produtos eleva a complexidade de descoberta de regras interessantes pela sua dimensão (muitas regras) e falta de representatividade (casos esparsos). A categorização dos dados permite reduzir as regras a dimensões interpretáveis e capazes de serem descartadas ou não por especialistas do domínio. Este método é chamado de mineração de regras de associação multinível.

As quatro maneiras mais comuns de lidar com dados em falta são:

- Apagar os registros – É de resolução fácil, mas pode eliminar valores de outros atributos que podem ser importantes;
- Substituir valores em falta pela média – Pode prejudicar os resultados introduzindo dados que não são reais;
- Substituir valores em falta por zeros – Os algoritmos entendem o valor como sendo um valor real prejudicando os resultados;
- substituir valores em falta pelo valor do vizinho mais próximo ou pela média dos k vizinhos mais próximos – Dependendo dos dados, este valor poderá estar corretamente aproximado ou completamente errado.

2.2.1 Minimização da dependência estatística

Neste trabalho, o método proposto utiliza informações estatísticas de alta ordem sobre os dados utilizados a fim de fornecer um espaço de projeção, que, em contraste com a PCA, preserva melhor as informações contidas na nuvem de dados (COSTA; CAMPOS; BARROS, 2011). Este método utiliza a codificação eficiente por meio da utilização da

analise de componentes independente, do inglês *independent component analisys* (ICA). O ICA usa de estatísticas de alta ordem sobre os dados de entrada assim, tem o potencial para melhorar a separabilidade de classes no espaço projetado (LUCENA et al., 2011).

2.2.1.1 *Independent Component Analysis (ICA)*

O ICA é uma técnica de projeção de subespaço amplamente utilizada, que projeta dados de um espaço de alta dimensão para um espaço de menor dimensão (KIM et al., 2005) (BARTLETT; MOVELLAN; SEJNOWSKI, 2002) (HYVÄRINEN; KARHUNEN; OJA, 2004) (HYVÄRINEN, 1999). Esta técnica é uma generalização do PCA que descorrelaciona as estatísticas de alta ordem para além das informações de segunda ordem. No presente estudo, as bases da ICA foram calculadas utilizando o algoritmo FastICA (HYVÄRINEN; KARHUNEN; OJA, 2004), enquanto que outros métodos, tais como Infomax (BARTLETT; MOVELLAN; SEJNOWSKI, 2002), a probabilidade máxima (HYVÄRINEN, 1999) e Jade (RIBEIRO et al., 2015a) (RIBEIRO et al., 2015b) pode também ser empregados.

Um dos algoritmos ICA mais amplamente utilizados é o algoritmo FastICA, proposto por Hyvärinen e Oja (HYVARINEN, 1999). Ele baseia-se na otimização de uma função de contraste que mede a não Gaussianidade da mistura. A popularidade do FastICA pode ser atribuída à sua simplicidade, facilidade de implementação e flexibilidade para escolher a função de não-linearidade (WEI, 2017).

Existem duas versões de algoritmos FastICA: FastICA de uma unidade e a FastICA simétrica. A versão de uma unidade do FastICA extrai uma fonte de cada vez até que todas as fontes sejam recuperadas. Para evitar que o algoritmo faça a convergência para a mesma fonte duas vezes, um procedimento adicional denominado *deflation* é necessário (DELFOSSE; LOUBATON, 1995). O Fastica de uma unidade tem uma desvantagem comum de todos os esquemas de separação sequencial de fontes: a propagação de erros durante a extração sucessiva de problemas com grande dimensionalidade. A versão simétrica do FastICA (OJA; YUAN, 2006) extrai todos os sinais de origem simultaneamente. Nesse trabalho foi utilizada o FastICA simétrica pois a mesma não sofre a desvantagem da propagação de erros e geralmente é mais estável.

2.3 Classificadores

A descrição de alguns do algoritmos mais utilizados em mineração de dados para bases de dados de saúde é realizada nessa seção.

2.3.1 Árvore

A árvore de decisão é uma representação matemática de um modelo com regras estruturadas numa hierarquia de classes ou valores (AZEVEDO; SANTOS, 2008) aplicáveis

a tarefas de classificação ou regressão. A representação gráfica assemelha-se a uma árvore natural sendo a representação mais comum invertida, como topo simboliza a raiz e na base encontramos as folhas.

A estrutura da árvore de decisão é composta por três elementos: Nós internos, folhas e ramos. A raiz da árvore é um nó interno. Os nós internos correspondem à avaliação de um atributo. Dos nós internos saem dois ou mais ramos que correspondem às possibilidades de resultados da avaliação do nó interno. Quando essa avaliação pode resultar num só resultado, é representado um nó terminal puro chamado de folha a que corresponde a classe.

O processo de indução é realizado por um algoritmo, normalmente o C4.5 (QUINLAN, 2014) ou o Ant-Tree-Miner (OTERO; FREITAS; JOHNSON, 2012) inspirado no funcionamento de uma colônia de formigas. O algoritmo gulosos, do inglês (*greedy*), constrói a árvore de cima para baixo, ou seja, da raiz para as folhas, tendo por base um conjunto de dados de treino devidamente classificados. Este fato quer dizer que a indução da árvore de decisão é uma aprendizagem supervisionada. O processo de seleção do atributo a colocar no nó interno é chamado de Método de Seleção do Atributo (YOO et al., 2012) e é o fator mais importante para a performance do modelo depois da qualidade dos dados do treino.

Existem várias técnicas de seleção do atributo, mas todas têm como objetivo encontrar aquele cujos valores melhor contribuem para dividir os registos de acordo com as classes. Essas técnicas, tentam encontrar o atributo em que a variabilidade dos valores mais se assemelhe à variabilidade das classes. Os critérios de seleção dos atributos são usados para calcular os mais significativos, isto é, os que mais contribuem para definir a classe (BORGES, 2016). Este processo continua tendo por base o novo nó, dividindo o conjunto de dados até que seja encontrada a classe predominante que dá origem à folha. Se o conjunto de dados possui muitos atributos e classes possíveis a árvore pode ficar demasiado complexa e prejudicar a performance do modelo.

Quando terminado o processo de indução, a árvore encontra-se ajustada aos dados oferecendo uma estrutura rígida para classificar exemplos não treinados ou simplesmente porque a estrutura contém mais do que o necessário. Para resolver estes problemas é necessário fazer a poda (*prunnings*) da árvore que é, no fundo, uma técnica que implica a eliminação de nós internos que não contribuem de forma decisiva para o resultado final (BORGES, 2016). A poda pode ser feita durante ou após a aprendizagem do modelo e baseia-se em cálculos de previsão de uma taxa de erro. A poda resulta em árvores menores com melhor potencial e precisão (QUINLAN, 2014) (HAN; PEI; KAMBER, 2011). Para realizar a poda de forma correta e simplificar estruturas complexas podem ser necessários muitos dados bem como a seleção dos mais apropriados.

As árvores de decisão são de construção simples e rápida, de representação compreensível pelos utilizadores e onde estes podem visualizar claramente os atributos que mais

contribuem para a definição da classe.

2.3.2 Máquina de vetores de suporte

Uma máquina de vetores de suporte, do inglês *support vector mahine* (SVM), é uma representação matemática de um modelo de classificação tipicamente binária. Os casos são representados num espaço e o classificador calcula então uma fronteira (híper-plano) que separa ambas as classes de forma equitativa. Para classificar novos casos, basta verificar em que lado da fronteira este é representado. Apesar de inicialmente só ser possível uma classificação através de uma função linear, ou seja, a fronteira entre classes ser uma linha reta, desenvolvimentos mais recentes tornam possível a classificação não-linear através de funções polinomiais e de base radial (BORGES, 2016).

A maior vantagem desta técnica encontra-se na capacidade de atingir bons resultados de classificação na generalidade dos casos de problemas de classes binárias: dos 21 testes realizados por (MEYER; LEISCH; HORNIK, 2003) a SVM encontrava-se entre os melhores em 19 deles. Este fato não deve ser confundido como sendo esta a melhor técnica de classificação pois o mesmo depende muito da estrutura dos conjuntos de dados.

A técnica SVM é muito dependente da função selecionada e do número de classes. Como as SVM são classificadores binários, os problemas que envolvam classes ternárias ou superiores são reduzidos a problemas de classes binárias tratados por associação de classificadores. Quando comparada com outras técnicas de classificação, a SVM tem um treino lento e requer significativos recursos computacionais (MEYER; LEISCH; HORNIK, 2003).

2.3.3 *k-Nearest Neighbours*

Os algoritmos da técnica *Nearest Neighbours* (NN) baseiam-se em medidas de distância entre objetos representados num plano (BORGES, 2016). Visto que a classificação se baseia em distância, estes algoritmos são ideais para atributos numéricos normalizados e não para atributos categóricos. A predição baseia-se na premissa de que objetos que estão perto são mais similares do que objetos mais distantes. O parâmetro k indica o número de objetos que são considerados na determinação da classe. Cada objeto que participa na vizinhança vota para a determinação da classe sendo esta obtida por maioria. É comum considerar pesos diferentes na votação com a proximidade dos objetos sendo que objetos mais perto têm maior contribuição que objetos mais afastados. Uma das vantagens destes algoritmos é o fato de serem *lazy*, isto é, não processarem dados na fase de treino, fazendo apenas memorização dos dados, e deixando todo o trabalho para a predição. Num modelo complexo com muita dimensionalidade, muitos atributos, a predição pode ser computacionalmente custosa na determinação da distância, recorrendo por isso a

subconjuntos de atributos relevantes. Os algoritmos desta técnica são incrementais pois a adição de novos objetos não obriga a recálculo do modelo.

2.3.4 Ensemble

A técnica de Ensemble é na prática uma associação de classificadores com o objetivo de minimizar a taxa de erro e assim permitir obter resultados melhores do que o faz uma técnica de forma individual. A técnica começa por construir diversos modelos classificadores, necessariamente diferentes uns dos outros.

- Conjunto de dados de treino – Cada classificador usará um conjunto de dados de treino diferente, criados através de diferentes métodos de amostragem. As abordagens mais conhecidas são as seguintes:
  - *Bootstrap Aggregation* (*Bagging*) – São gerados vários conjuntos de dados de treino com base no conjunto de treino original através de amostragem aleatória com reposição. Os exemplos do conjunto de dados de teste são classificados por todos os modelos sendo a classificação final obtida por voto majoritário, ou seja, a classe predominante no resultado de todos os classificadores.
  - *Boosting* – Gera um conjunto de dados de treino com base no treino anterior (e não no conjunto original) em que os casos mais difíceis de classificar tem seu peso aumentado para que apareçam com maior frequência na re-amostra e que os mais fáceis tem seu peso reduzido para que outros casos novos sejam escolhidos em seu lugar. O método de boosting mais conhecido é o *Adaptive Boosting* (Adaboost) (FREUND; SCHAPIRE, 1997) que usa um coeficiente de ponderação para cada classificador construído iterativamente calculado com base na taxa de erro e que é usado para determinação da classificação na votação final;
- Atributos – Os conjuntos de dados são criados através da seleção de atributos por métodos de análise da sua correlação, através de técnicas de segmentação, por recomendação de peritos ou mesmo de forma aleatória. A abordagem mais conhecida é a *Random Forests* (BREIMAN, 2001) que induz diversas Árvores de Decisão com base em conjuntos de treino gerados a partir do conjunto original que diferem nos atributos e que por isso geram diferentes modelos. Seu resultado é obtido por votação;
- Classes - Os conjuntos de dados de treino são criados para que as classes possam concorrer entre si (BORGES, 2016). As abordagens de combinação entre classes múltiplas mais conhecidas são as seguintes:

  - *One against all* (OAA, 1-R) – Cada classe concorre contra um grupo constituído pelas demais existindo tantos classificadores como classes. A votação final é obtida por maioria;
  - *One against one* (OAO, 1-1) – Cada classe concorre somente com outra. São criados tantos classificadores quantas forem as combinações possíveis de pares de classes. Neste caso os exemplos de outras classes são ignorados na indução do modelo. A determinação da classe final é decidida por voto maioritário;
  - *Error-Correcting Output Coding* (ECOC) (DIETTERICH; BAKIRI, 1995), Cada classe é codificada por uma *codeword* binária. São construídos tantos classificadores quantos os bits desta *codeword*. Cada classificador prevê um dos bits da *codeword* e a classificação final é obtida através da comparação da distância de Hamming (HAMMING, 1950) obtida e das *codewords* de cada classe, ganhando aquela em que a distância é menor, onde houver menos variação.

- Algoritmo de treino – Neste caso o que muda não são aos dados, mas sim aos algoritmos de indução dos modelos que vêm os seus parâmetros alterados e que por esse motivo resultam em classificadores diferentes. A classificação obtém-se por voto maioritário.

A diferença nos modelos classificadores fará com que se obtenha um resultado individual com maior precisão e confiabilidade. Um caso notável é o Adaboost que atinge normalmente resultados superiores a uma SVM individual (Morra, et al., 2010) (Situ, et al., 2010) (Douglas, et al.,2010) (Lopes, et al. 2011). O estudo de (Douglas, et al.,2010) revela ainda que a Random Forest melhorou em 1% o resultado do Adaboost. Existe investigação que mostra ainda a possibilidade de criar uma combinação de combinações de classificadores (Ensemble de Ensembles) que poderão ainda melhorar os resultados, contudo os resultados tendem para um limite inultrapassável independentemente do número de combinações realizado (Ahn, et al., 2007).

2.4 Trabalhos Correlatos

Muitos estudos têm contribuído para a melhoria da classificação da diabetes utilizando a base de dados indiana Pima. Dentre eles Lee e Wang (2011) obtiveram uma acurácia de 91.2% utilizando uma ontologia fuzzy de cinco camadas, sendo estas camadas, conhecimento, relação de grupo, domínio de grupo, relação pessoal e a camada de domínio pessoal, foi desenvolvido um sistema especialista fuzzy para descrever conhecimento com incerteza. Aplicando a ontologia fuzzy ao domínio do diabetes, a estrutura da ontologia do fuzzy para diabetes é definida para modelar o conhecimento da diabetes.

Patil, Joshi e Toshniwal (2010) propuseram um Modelo de Previsão Híbrido (HPM) com 92.38% que utiliza o algoritmo de *clustering* de K-means simples para validação dos rótulos das classes dos dados escolhidos (as instâncias classificadas incorretamente foram removidas, isto é, o padrão extraído dos dados originais) que foram então aplicados subsequentemente ao algoritmo de classificação para obter o conjunto de resultado.

Byeon, Rasheed e Doshi (2008) conseguiram 92.60% usando uma abordagem mista de algoritmo genético (GA) e seleção de protótipos, essa abordagem foi utilizada para melhorar a qualidade dos dados de treinamento com variável dependente ruidosa para classificação binária. O ruído reduz a precisão da classificação, interrompendo o conjunto de dados de treinamento e fazendo com que o classificador construa modelos incorretos. Nessa abordagem (GAPS) é usado um algoritmo genético (GA) para criar o conjunto de instâncias ruidosas suspeitas e seleção de protótipo (PS) para identificar o conjunto de instâncias reais com ruido.

Ribeiro et al. (2015a) obtiveram 98.7% nessa abordagem os dados de entrada foram submetidos a um processo de extração de características para criar uma nova representação dos dados originais com redundância mínima. A nova representação, de acordo com a extração baseada na codificação eficiente, é a entrada para o classificador SVM de uma classe que agrupa as duas classes.

Nesse Trabalho é realizado a validação da metodologia de (RIBEIRO et al., 2015a) com a expansão da base utilização para todos os pacientes da região nordeste.

3 METODOLOGIA

Nesse capitulo é apresentado a metodologia desenvolvida nesse trabalho. Ela é composta de cinco passo conforme Figura 1. Estes são:

1. Seleção;
2. Pré-processamento e limpeza;
3. Transformação dos dados;
4. Classificação;
5. Avaliação dos resultados.

Figura 1 – Diagrama da metodologia utilizada dividida em cinco passos: seleção, pré-processamento, transformação dos dados, classificação e avaliação dos resultados.

Fonte: Produzido pelo Autor.

No primeiro passo, seleção, será descrita a forma como foi realizada a aquisição da base de dados com as 19 variáveis clinicas discriminadas na Tabela 1. No segundo passo, pré-processamento e limpeza, os dados foram limitados apenas aos pertencentes a região nordeste, também foi realizado a remoção dados duplicados, com valores discrepantes ou não preenchidos, ao termino desse passo, apenas variáveis clínicas não invasivas restaram. No terceiro passo, transformação dos dados, os valores não numéricos foram transformados em valores numéricos e os dados foram projetados para um novo espaço de representação. No quarto passo, classificação, foi realizado o treino e teste de diferentes classificadores, no final desse passo, os dados dos pacientes foram discriminados como diabéticos ou não diabéticos. Finalizando com o quinto passo, é apresentada a métrica que foi utilizada para avaliar a performance dos classificadores.

O detalhamento dos passos utilizados são apresentados nas seções seguintes.

3.1 Seleção

A base de dados utilizada nesse projeto foi uma amostra da base de dados do SisHiperdia, e é composta por dados de pacientes divididos por unidade federativa, totalizando 13500 diabéticos e 13500 não diabeticos.

O software MATLAB foi utilizado para a aquisição dos dados em duas matrizes uma composta com dados de pacientes diabéticos(MD - Matriz de Diabéticos) e outra com não diabéticos(MND - Matriz de Não Diabéticos). Cada matriz contém 13500 linhas ou observações, representando cada paciente e 19 colunas, representado as variáveis clínicas.

No passo seguinte, pré-processamento e limpeza, ocorreu a retirada de todas as variáveis invasivas, e realizado o agrupamento dos pacientes por região.

3.2 Pré-processamento e Limpeza

As matrizes MD e MND, obtidas no passo anterior, passaram pelo pré-processamento e limpeza. Nessa etapa as seguinte operações foram realizadas:

a) Remoção dos dados de pacientes não pertencentes a unidades federativas da região nordeste;

b) Remoção dos dados discrepantes, isto é, todos os dados fora do intervalo de aceitação da Tabela 2;

c) Remoção de todas as variáveis clínicas não presentes na Tabela 3;

d) Remoção de todas as linhas com dados faltantes.

Ao termino dessa etapa foi obtido duas matrizes de dados alvo(MDA), uma com os dados de pacientes diabéticos(MDAD) e outra de não diabéticos(MDAND) com 912 e 1044 linhas respectivamente, totalizando 1956 pacientes.

Tabela 2 – Intervalo de aceitação para os dados dos pacientes.

Variável	**Diabéticos**	**Hipertensos**
Idade(anos)	>18 e <65	>18 e <65
PAS (mmHg)	>70 e < 140	$\geq$ 140
PAD (mmHg)	>40 e < 90	$\geq$ 90
Cintura(cm)	>40 e < 220	>40 e < 220
Peso(kg)	>40 e < 220	>40 e < 220
Altura(cm)	>90 e < 220	>90 e < 220
Glicemia em jejum(mg/dl)	$\geq$ 126 e < 800	>70 e < 126

Fonte: Produzido pelos autores.

Tabela 3 – Variáveis clinicas utilizadas.

Número	**Marcador**
2	Idade
3	Sexo
4	PAS (mmHg)
5	PAD (mmHg)
6	Cintura (cm)
7	Peso (kg)
8	Altura (cm)
10	Antecedentes familiares
11	Tabagismo
12	Sedentarismo
13	Sobrepeso
14	Infarto

Fonte: Produzido pelos autores.

Os intervalos de valores apresentados na Tabela 2 serviram para minimizar os ruídos nos dados uma vez que eram encontrados valores como 2 mg/dl para glicemia em jejum, pesos inferiores a 20 kg, alturas inferiores a 40 centímetros, PAS maiores que 400 mmHg, esses valores não são humanamente possíveis, logo devem ser removidos para que não prejudiquem o processo de classificação.

As variáveis clínicas utilizadas nesse trabalho são descritas na Tabela 3, onde foram removidas as variáveis invasivas, além das seguintes variáveis: Outras coronariopatias, AVC, pé diabético e amputação.

Com a obtenção das matrizes MDAD e MDAND, foi dado inicio a próxima etapa da metodologia, transformação dos dados. A próxima seção descreve em detalhes seu passos.

3.3 Transformação dos Dados

Após a etapa de pré-processamento e a obtenção das matrizes MDAD e MDAND, as mesmas passaram pela etapa de transformação de dados essa etapa se dividiu em cinco partes:

a) Converter todos os dados textuais para numéricos;

b) Normalização das matrizes MDAD e MDAND;

c) Branqueamento da matriz MDAD;

d) Obtenção da matriz de mistura(MM) da matriz MDAD usando FastICA;

e) Obtenção da matriz de nova representação (MNR) após projetar as matrizes MDAD e MDAND usando a matriz MM.

Na primeira parte foi convertido todos os dados textuais para numéricos, nesta parte o sexo do paciente foi alterado de feminino (F) para 0 e de masculino (M) para 1. Em seguida as variáveis binarias 10, 11, 12, 13 e 14 foram modificadas de não (N) para 0 e de sim (S) para 1. Após essa transformação todos os dados das matrizes alvo tornaram-se do tipo numérico.

A segunda parte consistiu a normalização das matrizes de dado alvo, essa parte foi realizada para reduzir o custo computacional da próxima etapa.

Em seguida, foi branqueada a matriz MDAD, essa matriz branqueada(MDADB) foi então passada pelo algoritmo FastICA, para obtenção da Matriz de Mistura de Diabéticos (MMD).

Em seguida as matrizes MDAD e MDAND foram projetadas para um novo domínio, utilizando a matriz de mistura MMD e obtendo a matriz de nova representação(MNR). Ela foi normalizada gerando a matriz MNRN, após isso foi adicionada uma coluna a matriz MNRN com a discriminação da classe para a etapa de treino dos classificadores, os pacientes diabéticos com 1 e não diabéticos 0, criando assim a matriz para classificação(MPC). O algoritmo utilizado nessa parte está presente no Apêndice A.

3.4 Classificadores

A quarta etapa da metodologia consistiu em treinar os classificadores com os dados da matriz MPC obtida na seção anterior.

O classificadores utilizados foram treinado segundo seus tipos, essa sessão detalha o processo de treinamento.

3.4.1 Configuração do Classificador

3.4.1.1 Árvore

Foram utilizados três classificadores do tipo árvore:

a) Simples

b) Média

c) Complexa

Os três classificadores do tipo árvore foram configurados alterando o número de divisões realizadas pelo algoritmo conforme apresentado nos Algoritmos 3.1, 3.2 e 3.3,

esses trechos são parte do código desenvolvido nesse projeto, o código inteiro está presente no Apêndice B. A descrição dos parâmetros utilizados são discriminados na Tabela 4.

Tabela 4 – Parâmetros para configuração do Classificador Árvore.

Parâmetro	**Descrição**
SplitCriterion	Especifica o critério de separação.
MaxNumSplits	Número máximo de divisões de decisão (ou nós de ramificação).
ClassNames	Nomes das classes usadas para treinamento.

Fonte: Produzido pelos autores.

Algoritmo 3.1 – Algoritmo responsável pela especificação do classificador Árvore Simples.

```
% Especificacao da configuracao do classificador arvore simples
classificationTreeSimple = fitctree(predictors, response, ...
    'SplitCriterion', 'gdi', 'MaxNumSplits', 4, ...
    'ClassNames', [0; 1]);
```

Algoritmo 3.2 – Algoritmo responsável pela especificação do classificador Árvore Média.

```
% Especificacao da configuracao do classificador arvore media
classificationTreeMedium = fitctree(predictors, response, ...
    'SplitCriterion', 'gdi', 'MaxNumSplits', 20, ...
    'ClassNames', [0; 1]);
```

Algoritmo 3.3 – Algoritmo responsável pela especificação do classificador Árvore Complexa.

```
% Especificacao da configuracao do classificador arvore complexa
classificationTreeComplex = fitctree(predictors, response, ...
    'SplitCriterion', 'gdi', 'MaxNumSplits', 100, ...
    'ClassNames', [0; 1]);
```

Tabela 5 – Configuração dos Classificadores Árvore: Simples, Média e Complexa.

Algoritmo	**MaxNumSplits**
Simples	4
Média	20
Complexa	100

Fonte: Produzido pelos autores.

A Tabela 5 apresenta a diferença entres as três variantes do tipo árvore utilizada em relação ao número máximo de divisões possíveis.

3.4.1.2 Máquinas de Vetores de Suporte

Foram utilizados seis classificadores de Máquinas de Vetores de Suporte:

a) *Linear*

b) *Quadrática*

c) *Cúbica*

d) *Gaussiana fina*

e) *Gaussiana média*

f) *Gaussiana larga*

Os seis classificadores são configurados alterando os parâmetros *KernelFunction*, *PolynomialOrder* e *KernelScale*. A descrição dos parâmetros utilizados podem ser encontrados na Tabela 6.

Tabela 6 – Parâmetros para configuração do Classificador SVM.

Parâmetro	**Descrição**
KernelFunction	Função Kernel usada para calcular a matriz de Gram.
PolynomialOrder	Ordem do polinomio da função kernel.
Standardize	Normaliza os dados preditores.
KernelScale	O software divide todos os elementos da matriz preditora X pelo valor de KernelScale. Então, o software aplica a norma do kernel apropriada para calcular a matriz de Gram.
ClassNames	Nomes das classes usadas para treinamento.

Fonte: Produzido pelos autores.

A Tabela 7 sumariza a configuração dos classificadores SVM Linear, Quadrática e Cúbica, que são apresentadas nos Algoritmos 3.4, 3.5 e 3.6, esses trechos são parte do código desenvolvido nesse projeto, o código inteiro está presente no Apêndice C. Nela são especificadas as funções kernel e as ordens polinomiais utilizadas para a configuração dos três classificadores.

Algoritmo 3.4 – Algoritmo responsável pela especificação do classificador SVM Linear.

```
15 % Esse trecho especifica a configuracao do classificador svm
       linear
16 classificationLinearSVM = fitcsvm(predictors, response, ...
17     'KernelFunction', 'linear', 'PolynomialOrder', [], ...
18     'KernelScale', 'auto', 'Standardize', true, ...
19     'ClassNames', [0; 1]);
```

Tabela 7 – Configuração dos Classificadores SVM: Linear, Quadrática, Cúbica.

Algoritmo	**KernelFunciton**	**PolynomialOrder**
Linear	Linear	
Quadrática	Polinomial	2
Cúbica	Polinomial	3

Fonte: Produzido pelos autores.

Algoritmo 3.5 – Algoritmo responsável pela especificação do classificador SVM Quadrática.

```
% Esse trecho especifica a configuracao do classificador svm
    quadratica
classificationQuadraticSVM = fitcsvm(predictors, response, ...
    'KernelFunction', 'polynomial', 'PolynomialOrder', 2, ...
    'KernelScale', 'auto', 'Standardize', true, ...
    'ClassNames', [0; 1]);
```

Algoritmo 3.6 – Algoritmo responsável pela especificação do classificador SVM Cúbica.

```
% Esse trecho especifica a configuracao do classificador svm
    cubica
classificationCubicSVM = fitcsvm(predictors, response, ...
    'KernelFunction', 'polynomial', 'PolynomialOrder', 3, ...
    'KernelScale', 'auto', 'Standardize', true, ...
    'ClassNames', [0; 1]);
```

A Tabela 8 sumariza a configuração dos classificadores SVM Gaussianas: Fina, média e larga. Estes são apresentadas nos Algoritmos 3.7, 3.8 e 3.9. Nela são especificados a função kernel e os parâmetros KernelScale utilizados para a configuração dos três últimos classificadores.

Tabela 8 – Configuração dos Classificadores SVM Gaussianos: Fina, Média, Larga.

Algoritmo	**KernelFunction**	**KernalScale**
Fina	Gaussian	0.87
Média	Gaussian	3.5
Larga	Gaussian	14

Fonte: Produzido pelos autores.

Algoritmo 3.7 – Algoritmo responsável pela especificação do classificador SVM Gaussiana fina.

```
% Esse trecho especifica a configuracao do classificador svm gaussiana fina
classificationFineGaussianSVM = fitcsvm(predictors, response, ...
    'KernelFunction', 'gaussian', 'PolynomialOrder', [], ...
    'KernelScale', 0.87, 'Standardize', true, ...
    'ClassNames', [0; 1]);
```

Algoritmo 3.8 – Algoritmo responsável pela especificação do classificador SVM Gaussiana média.

```
% Esse trecho especifica a configuracao do classificador svm gaussiana media
classificationMediumGaussianSVM = fitcsvm(predictors, response, ...
    'KernelFunction', 'gaussian', 'PolynomialOrder', [], ...
    'KernelScale', 3.5, 'Standardize', true, ...
    'ClassNames', [0; 1]);
```

Algoritmo 3.9 – Algoritmo responsável pela especificação do classificador SVM Gaussiana larga.

```
% Esse trecho especifica a configuracao do classificador svm gaussiana larga
classificationCoarseGaussianSVM = fitcsvm(predictors, response, ...
    'KernelFunction', 'gaussian', 'PolynomialOrder', [], ...
    'KernelScale', 14, 'Standardize', true, ...
    'ClassNames', [0; 1]);
```

3.4.1.3 K Vizinhos Mais Próximos

Foram utilizado seis classificadores de k vizinhos mais próximos:

a) Fino

b) Médio

c) Cúbica

d) Largo

e) Cosseno

f) Ponderado

Os seis classificadores são configurados alterando alguns dos parâmetros da Tabela 9. A descrição dos parâmetros utilizados podem ser encontrados na Tabela 9.

Tabela 9 – Parâmetros para configuração do Classificador SVM.

Parâmetro	**Descrição**
Distance	Define a distancia utilizada. Os nomes de métrica de distância permitidos dependem da escolha de um método de busca de vizinho.
Exponent	Expoente da distância de Minkowski.
NumNeighbors	Número de vizinhos mais próximos para encontrar.
DistanceWeight	Função de ponderação da distância
Standarlize	Normaliza os dados preditores.
ClassNames	Nomes das classes usadas para treinamento.

Fonte: Produzido pelos autores.

A Tabela 10 sumariza a configuração dos classificadores KNN fino, médio e largo, que são apresentadas nos Algoritmos 3.10, 3.11 e 3.12, esses trechos são parte do código desenvolvido nesse projeto, o código inteiro está presente no Apêndice D. Nela são especificadas as funções kernel e as ordens polinomiais utilizadas para a configuração dos três classificadores.

Algoritmo 3.10 – Algoritmo responsável pela especificação do classificador KNN fino.

```
15 % Esse trecho especifica a configuracao do classificador KNN
       fino
16 classificationKNNFine = fitcknn(predictors, response, ...
17     'Distance', 'Euclidean', 'Exponent', [], ...
18     'NumNeighbors', 1, 'DistanceWeight', 'Equal', ...
19     'Standardize', true, 'ClassNames', [0; 1]);
```

Algoritmo 3.11 – Algoritmo responsável pela especificação do classificador KNN médio.

```
21 % Esse trecho especifica a configuracao do classificador KNN
       medio
22 classificationKNNMedium = fitcknn(predictors, response, ...
23     'Distance', 'Euclidean', 'Exponent', [], ...
24     'NumNeighbors', 10, 'DistanceWeight', 'Equal', ...
25     'Standardize', true, 'ClassNames', [0; 1]);
```

Algoritmo 3.12 – Algoritmo responsável pela especificação do classificador KNN largo.

```
% Esse trecho especifica a configuracao do classificador KNN
    largo
classificationKNNCoarse = fitcknn(predictors, response, ...
    'Distance', 'Euclidean', 'Exponent', [], ...
    'NumNeighbors', 100, 'DistanceWeight', 'Equal', ...
    'Standardize', true, 'ClassNames', [0; 1]);
```

Tabela 10 – Configuração dos Classificadores KNN: Fine, Medium, Coarse.

Algoritmo	**Distance**	**NumNeighbors**
Fina	Euclidean	1
Média	Euclidean	10
Larga	Euclidean	100

Fonte: Produzido pelos autores.

A Tabela 11 sumariza a configuração dos classificadores SVM Gaussianas: Fina, média e larga. Estes são apresentadas nos Algoritmos 3.13, 3.14 e 3.15. Nela são especificados a função kernel e os parâmetros KernelScale utilizados para a configuração dos três últimos classificadores.

Algoritmo 3.13 – Algoritmo responsável pela especificação do classificador KNN Cúbica.

```
% Esse trecho especifica a configuracao do classificador KNN
    cubico
classificationKNNCubic = fitcknn(predictors, response, ...
    'Distance', 'Minkowski', 'Exponent', 3, ...
    'NumNeighbors', 10, 'DistanceWeight', 'Equal', ...
    'Standardize', true, 'ClassNames', [0; 1]);
```

Algoritmo 3.14 – Algoritmo responsável pela especificação do classificador KNN Cosseno.

```
% Esse trecho especifica a configuracao do classificador KNN
    coseno
classificationKNNCosine = fitcknn(predictors, response, ...
    'Distance', 'Cosine', 'Exponent', [], ...
    'NumNeighbors', 10, 'DistanceWeight', 'Equal', ...
    'Standardize', true, 'ClassNames', [0; 1]);
```

Algoritmo 3.15 – Algoritmo responsável pela especificação do classificador KNN Ponderado.

```
% Esse trecho especifica a configuracao do classificador KNN
    ponderado
classificationKNNWeighted = fitcknn(predictors, response, ...
    'Distance', 'Euclidean', 'Exponent', [], ...
    'NumNeighbors', 10, 'DistanceWeight', 'SquaredInverse', ...
    'Standardize', true, 'ClassNames', [0; 1]);
```

Tabela 11 – Configuração dos Classificadores KNN: Cúbico, Cosseno, Ponderado.

Algoritmo	**Distance**	Exponent	**NumNeighbors**	DistanceWeight
Cúbica	Minkowski	3	10	Equal
Cossena	Cosine		10	Equal
Ponderada	Euclidean		10	SquaredInverse

Fonte: Produzido pelos autores.

3.4.1.4 Ensemble

Foram utilizado cinco classificadores do tipo ensemble, estes foram:

a) Árvore *Bagged*

b) Subespaço KNN

c) Subespaço Discriminante

d) Árvore *RUSBoosted*

e) Árvore *Boosted*

Os cinco classificadores são configurados alterando alguns dos parâmetros da Tabela 12. A descrição dos parâmetros utilizados podem ser encontrados na Tabela 12. A configuração do classificador ensemble está presente no Apêndice E

Após a etapa de treinamento foi iniciado a etapa de treino e teste dos classificadores.

3.4.2 Treino e Testes

Após as configurações dos classificadores foi realizado o treino dos mesmo, para o treino e teste foi utilizado a técnica de validação cruzada, *k-fold*, com k igual a vinte.

Tabela 12 – Parâmetros para configuração do Classificador SVM.

Parâmetro	**Descrição**
Subspace	Função Kernel usada para calcular a matriz de Gram.
Type	Ordem do polinomio da função kernel.
ClassNames	Nomes das classes usadas para treinamento.

Fonte: Produzido pelos autores.

3.5 Avaliação dos resultados

Nos testes foram utilizados a validação cruzada. Esta técnica de particionamento que divide a matriz MPC em k partes disjuntas. Para a avaliar o desempenho dos classificadores quanto sua capacidade de diferenciação entre diabéticos e não diabéticos foram utilizadas as matrizes de confusão, obtida de cada teste da etapa anterior. Como critérios de avaliação de performance foram utilizados a sensibilidade, especificidade e acurácia. Estes termos possuem variáveis que vamos relacionar para melhor entendimento destes, como veremos a seguir. As variáveis são:

- Verdadeiro Positivo(VP): Diagnóstico do paciente classificado corretamente como diabético;
- Falso Positivo(FP): Diagnóstico do não-diabético classificado como diabético;
- Verdadeiro Negativo(VN): Diagnóstico do paciente classificado corretamente como não-diabético;
- Falso Negativo(FN): Diagnóstico do diabético classificado como não-diabético.

A sensibilidade, Equação 3.1, indica quão bom é o classificador para identificar os pacientes diabéticos e é definida por:

$$sensibilidade = \frac{VP}{VP + FN} \tag{3.1}$$

A especificidade, Equação 3.2, indica quão bom é o classificador para identificar os pacientes não-diabéticos e é definida por:

$$especificidade = \frac{VN}{VN + FP} \tag{3.2}$$

A acurácia, Equação 3.3, é uma taxa calculada pelo número de instâncias bem classificadas sobre o valor de instâncias total e é dada por:

$$acurácia = \frac{VP + VN}{VP + FN + VN + FP} \tag{3.3}$$

Os resultados obtidos nessa etapa são apresentado no Capítulo 4 e discutidos no Capítulo 5.

4 RESULTADOS

Nesse capítulo será apresentado os resultados obtidos para a região nordeste utilizando o metodologia descrito no Capítulo 3. A sessões seguintes apresenta os resultados obtidos para cada classe de classificadores utilizado. Salvo a ultima seção desse capítulo, on os melhores classificadores das seções anteriores são comparados.

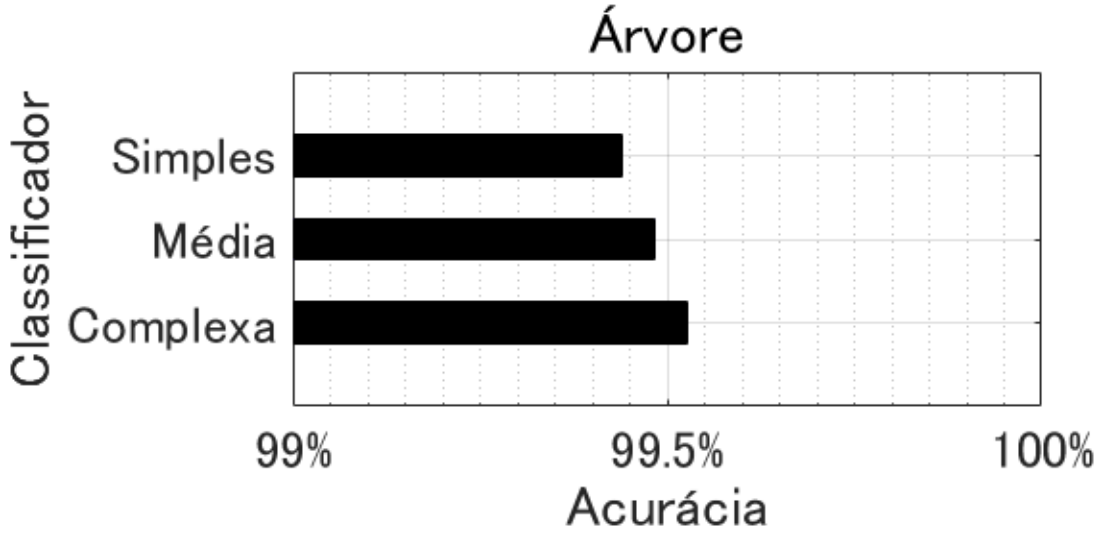
4.1 Árvore

A Figura 2 apresenta os resultados obtidos para os classificadores do tipo árvore utilizados, os valores de acurácia, especificidade e sensibilidade estão na Tabela 13.

Os valores obtidos para as três variantes do classificador do tipo árvore foram apresentados na Tabela 13, os tipos usados foram: Simples, média e complexa. E serão discutidos nessa seção, a relação entre as variantes e o números de divisões (nós) foi apresentada na Tabela 5.

Esse classificador apresentou uma melhora na acurácia e sensibilidade com o aumento no número máximo de divisões. Os valores de acurácia para árvores simples, média e complexa foram de 99.44%, 99.48% e 99.53% respectivamente. Os valores de sensibilidade para árvores simples, média e complexa foram de 99.23%, 99.43% e 99.52% respectivamente. Contudo, o aumento de 4 para 20 divisões máximas piorou a performance do classificador em relação a especificidade, caindo de 99.61% para 99.53% este ultimo se manteve constante com o aumento no número máximo de nós de 20 para 100.

Figura 2 – Resultado Nordeste



Fonte: Autor.

Tabela 13 – Resultados Nordeste Arvores

Classificador	**Acurácia**	**Sensibilidade**	**Especificidade**
Árvore-Complexa	99.53%	99.52%	99.53%
Árvore-Média	99.48%	99.43%	99.53%
Árvore-Simples	99.44%	99.23%	99.61%

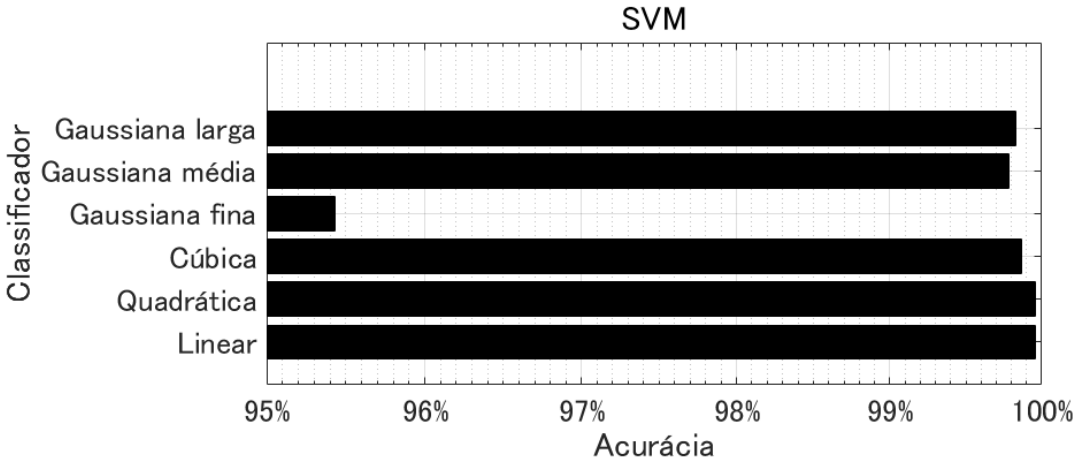
Fonte: Autor.

Dentre os classificadores do tipo árvore utilizados nessa pesquisa o melhor foi o classificador árvore complexa.

4.2 SVM

Os resultados para os classificadores utilizando SVM são apresentado na Figura 3 valores de acurácia, especificidade e sensibilidade estão na Tabela 14.

Figura 3 – Resultado Nordeste SVM



Fonte: Autor.

Tabela 14 – Resultados Nordeste SVM

Classificador	**Acurácia**	**Sensibilidade**	**Especificidade**
SVM-Linear	99.96%	99.90%	100.00%
SVM-Quadrática	99.96%	99.90%	100.00%
SVM-Cúbica	99.87%	99.71%	100.00%
SVM-Gaussiana fina	95.43%	89.86%	100.00%
SVM-Gaussiana média	99.78%	99.52%	100.00%
SVM-Gaussiana larga	99.83%	99.62%	100.00%

Fonte: Autor.

Os valores obtidos para as seis variantes do classificador SVM apresentados na Tabela 14 serão discutidos nessa seção. Para os classificadores SVM foram utilizados três tipos de função kernel: Linear, polinomial e gaussiana.

Apenas uma função kernal linear foi utilizada, a mesma obteve 99.96%, 99.90% e 100% de acurácia, sensibilidade e especificidade, respectivamente.

As funções de kernel polinomial utilizadas foram de segunda e terceira ordem como apresentado na Tabela 7. Essas funções obtiveram o mesmo valor de especificidade 100%. Contudo, diferentes valores para acurácia e sensibilidade. Com a função kernel quadrática foram obtidos 99.96% e 99.90% e para acurácia e sensibilidade, enquanto a função kernel cúbica obteve 99.87% e 99.71% para acurácia e sensibilidade. Dado a natureza do problema que se deseja classificar um classificador com alta a acurácia e alta sensibilidade é mais adequado. Isso torna a variante com função kernel quadrática melhor na tarefa que a com função kernel cúbica.

As três variantes utilizadas, com função kernel gaussiana, obtiveram melhores valores de acurácia e especificidade, a medida em que a ordem do polinômio crescia. Aumentando de 95.43% e 89.86% para 99.83% e 99.62%, em acurácia e especifidade, respectivamente, quando ordem polinomial foi de 0.87 para 14. Dentre as variantes gaussianas a gaussiana larga apresentou melhores resultados.

Para seleção da melhor função kernal, foram comparados os melhores resultados dentre as três funções kernel utilizadas. Essa comparação é sumarizada na Tabela 15.

Tabela 15 – Comparação entre as melhores variantes das funções kernel utilizadas: Linear, polinomial e gaussiana.

Classificador	**KernelFunciton**	**PolynomialOrder**	**Acurácia**	**Sensibilidade**	**Especificidade**
SVM-Linear	Linear		99.96%	99.90%	100.00%
SVM-Quadrática	Polinomial	2	99.96%	99.90%	100.00%
SVM-Gaussiana larga	Gaussian	14	99.83%	99.62%	100.00%

Fonte: Autor.

As variante do classificador SVM linear e quadrática apresentaram os melhores valores dentre todas as variante comparadas, com 99.96%, 99.90% e 100.00% de acurácia, sensibilidade e especificidade, respectivamente.

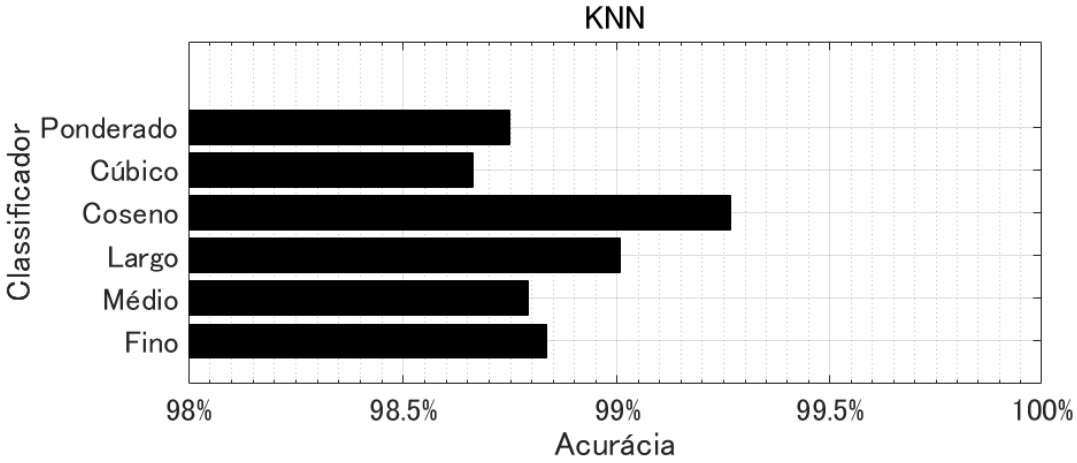
4.3 KNN

A Figura 4 apresenta os resultados obtidos para os classificadores KNN utilizados, os valores de acurácia, especificidade e sensibilidade estão na Tabela 16.

Os valores obtidos para as seis variantes do classificador KNN apresentados na Tabela 16, serão discutidos nessa seção. Para os classificadores KNN foram utilizados variações na quantidade de vizinhos, no calculo da distância e no peso da distância.

O aumento no número de vizinhos de 1 para 10 no classificador KNN apresentou uma piora nos valores de acurácia e sensibilidade de 98.84% e 98.47% para 98.79% e 97.80%,

Figura 4 – Resultado Nordeste KNN



Fonte: Autor.

Tabela 16 – Resultados Nordeste Arvores KNN

Classificador	**Acurácia**	**Sensibilidade**	**Especificidade**
KNN-Fino	98.84%	98.47%	99.14%
KNN-Médio	98.79%	97.80%	99.61%
KNN-Cúbica	99.01%	98.09%	99.76%
KNN-Largo	99.27%	98.56%	99.84%
KNN-Coseno	98.66%	97.61%	99.53%
KNN-Ponderado	98.75%	97.89%	99.45%

Fonte: Autor.

contudo o aumento de 10 para 100 vizinhos minimizou as percas para esse classificador aumento os valores de acurácia e sensibilidade para 99.01%, 98.09%. A melhor variante nessa categoria foi o KNN fino com 10 vizinhos.

Alterando a forma de calculo da distancia, foram obtidos uma melhora na performance usando uma distancia cosseno quando comparada com uma distancia do tipo cúbica. A variante cúbica obteve acurácia e sensibilidade de 98.66% e 97.61%. Enquanto a coseno obteve tando acurácia quanto sensibilidade superiores 99.27% e 98.56%, respectivamente.

A variante Pondera obteve 98.75% e 97.89% para acurácia e especificidade, respectivamente.

Para selecionar a melhor variante do classificador KNN, foram comparados os melhores resultados dentre os tipos anteriormente explicados. Essa comparação é sumarizada na Tabela 17.

A melhor variante KNN foi a KNN cosseno com 99.27% , 98.56% e 99.84% de acurácia, sensibilidade e especificidade, respectivamente.

Tabela 17 – Comparação entre as melhores variantes do classificador KNN.

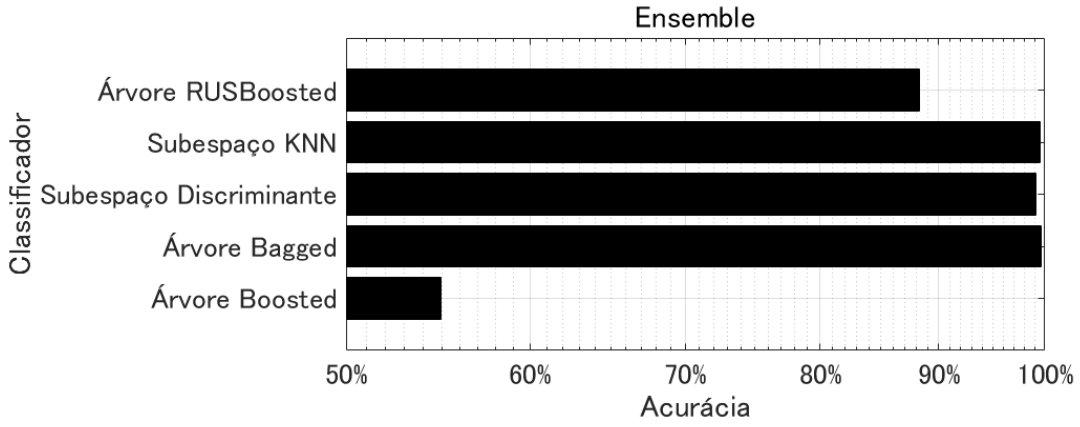
Classificador	**Distance**	**NumNeighbors**	**DistanceWeight**	**Acurácia**	**Sens.**	**Espec.**
KNN-Ponderada	Euclidean	10	SquaredInverse	98.75%	97.89%	99.45%
KNN-Fina	Euclidean	10	Equal	98.84%	98.47%	99.14%
KNN-Coseno	Cosine	10	Equal	99.27%	98.56%	99.84%

Fonte: Autor.

4.4 Ensemble

Os resultados para os classificadores utilizando Ensemble são apresentado na Figura 5 valores de acurácia, especificidade e sensibilidade estão na Tabela 18.

Figura 5 – Resultado Nordeste Ensemble



Fonte: Autor.

Tabela 18 – Resultados Nordeste Ensemble

Classificador	**Acurácia**	**Sensibilidade**	**Especificidade**
Ensemble-Árvore *Boosted*	54.94%	0.00%	100.00%
Ensemble-Árvore *Bagged*	99.61%	99.33%	99.84%
Ensemble-Subespaço Discriminante	99.14%	98.76%	99.45%
Ensemble-Subespaço KNN	99.48%	99.04%	99.84%
Ensemble-Árvore *RUSBoosted*	88.31%	74.55%	99.61%

Fonte: Autor.

Os valores obtidos para as cinco variantes do classificador ensemble apresentados na Tabela 18 serão discutidos nessa seção. Três tipos de variantes para os classificadores ensemble foram utilizados: Árvores, KNN e discriminante.

Dentre as variantes ensemble do tipo árvore, a árvore *bagged* obteve a maior acurácia, 99.61%, as outras duas variantes não se demonstraram eficientes na tarefa de classificação de pacientes diabéticos. A árvore *RUSBoosted* obteve apenas 88.31% de sensibilidade e a

árvore *boosted* falhou em todas as tentativas de diagnósticos de pacientes diabéticos. Desta forma o melhor classificador ensemble para as variantes do tipo árvores foi a *bagged*.

A variante KNN obteve 99.48%, 99.04% e 99.94% para acurácia, sensibilidade e especificidade, respectivamente.

Finalizando com a variante discriminante, seus valores de acurácia, sensibilidade e especificidade foram 99.14%, 98.76% e 99.45%, respectivamente.

Para a escolha da melhor variante do classificador ensemble, foram comparados os melhores resultados dentre os tipos anteriormente explicados. Essa comparação é sumarizada na Tabela 19.

Tabela 19 – Comparação entre as melhores variantes do classificador ensemble.

Classificador	**Acurácia**	**Sens.**	**Espec.**
Árvore *Bagged*	99.61%	99.33%	99.84%
Subespaço KNN	99.48%	99.04%	99.84%
Subespaço discriminante	99.14%	98.76%	99.45%

Fonte: Autor.

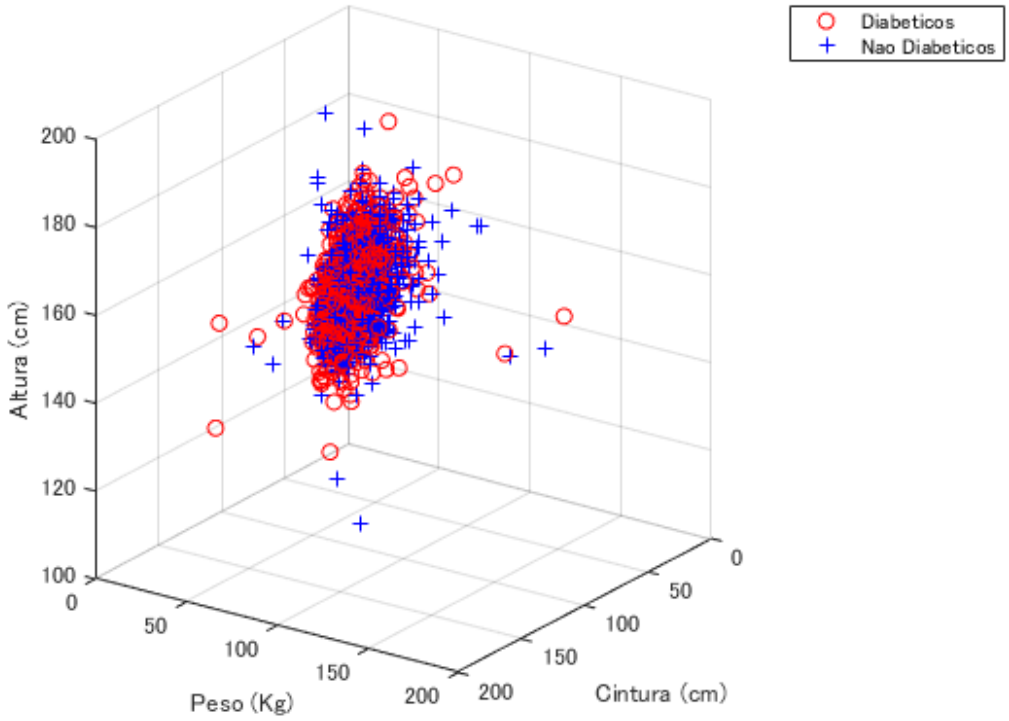
A melhor variante para o classificador Ensemble foi a árvore *Bagged* com 99.61%, 99.33% e 99.84% de acurácia, sensibilidade e especificidade, respectivamente.

5 ANÁLISE E DISCUSSÕES

Nesse capítulo serão analisados e discutidos os resultados apresentados no Capítulo 4.

A etapa de transformação dos dado foi responsável pela melhora da separabilidade entre as classes de pacientes. A Figura 6 mostra a disposição espacial em três dimensões de três variáveis clínicas: Altura, cintura e peso, após a etapa de pré-processamento e antes da etapa de transformação dos dados. Nela pode-se perceber que os pacientes diabéticos e não diabéticos apresentam-se com alta entropia.

Figura 6 – Disposição espacial em três dimensões de três variáveis clínicas, altura, cintura e peso, antes da utilização do ICA.

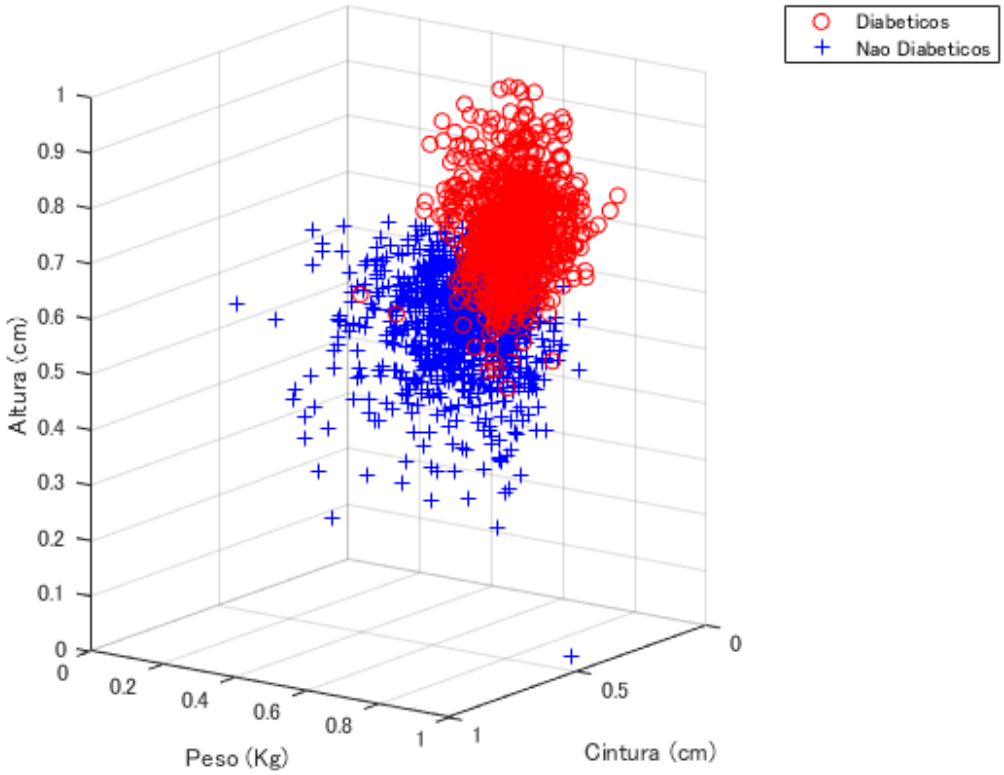


Fonte: Autor.

Após a etapa de transformação dos dados, Figura 7, percebeu-se uma redução na entropia entre os pacientes e com isso uma melhora separação espacial entre as classes, essa redução é visível quando comparamos a Figura 7 com a Figura 6.

Os dados dos pacientes diabéticos após etapa de transformação dos dados, apresentaram-se mais contidos a uma região do subespaço, já os dados dos pacientes não diabéticos apresentam-se mais dispersos, como mostrado na Figura 7. O mesmo comportamento

Figura 7 – Disposição espacial em três dimensões de três variáveis clínicas, altura, cintura e peso, após a utilização do ICA.



Fonte: Autor.

também foi percebido com outros conjuntos de três variáveis analisadas no espaço tridimensional. Desta forma, deve também ser percebido para dimensões mais altas. Esse comportamento de agrupamento dos pacientes diabéticos foi analisado e explorado por (RIBEIRO et al., 2015a).

A redução na entropia entre as classes, diabéticos e não diabéticos, melhorou as performances de todas as variantes de classificadores utilizados, salvo a variante do classificador ensemble: Árvore *Boosted*. Este caso obteve apenas uma alta especifidade pois, na fase de treinamento ocorreu um erro de classificação e o AdaBoostM1 não conseguiu realizar as convoluções e assim, não aprendendo a diferencias as classes. No caso do *Boosted* para essa aplicação o fato do mesmo gerar o conjunto de dados de treino com base no treino anterior prejudicou o calculo do coeficiente de ponderação que é baseado na taxa de erro. Contudo o classificador do tipo ensemble variante *Bagged* obteve o melhor resultado tendo o conjunto de dados de treino com base no conjunto de treino original através de amostragem aleatória com reposição.

O aumento no número de vizinhos para o classificador KNN melhorou a especificidade com o deslocamento do centroide dos não diabéticos. Esse deslocamento ajudou a

melhorar a especificidade de 99.14% para 99.76%.

Os cinco melhores resultados obtidos no capítulo anterior são do classificador SVM. Este tipo de classificador foi o quê obteve melhores resultados em comparação com os demais, devido a sua alta complexidade matemática em operações algébricas, como produto interno (transformações de espaço) e convergência, utilizando multiplicadores de lagrange, que proporcionam robustez ao método, pois os *clusters* são identificados de forma não-liner e reconvertidos a um espaço linear para simplificar a problemática da alta dimensionalidade da base de dados Hiperdia. A melhora nos resultados é diferenciada pela escolha da função kernel apropriada. Neste caso as melhores foram as linear e a quadrática.

Este tipo de classificador obteve o terceiro melhor resultado dentre os quatro tipo de classificadores utilizados e os piores resultados na variante simples, com uma acurácia de 99.44%. Isto devido ao particionamento retangular inerente do classificador. Apesar das classes analisadas nessa pesquisa aparentarem não está contida em uma região retangular e se em outra representação geométrica multidimensional esse classificar obteve bons resultados, com alto poder de aprendizado no caso dos não-diabéticos (especificidade), a falta de poder de aprendizado dos diabéticos (sensibilidade) fez decair a relação completa deste, a acurácia.

6 CONCLUSÕES

O estudo começou com a seguinte premissa:

É possível identificar diabetes em indivíduos etnicamente heterogêneos, mesmo sem recorrer a testes bioquímicos?

Conforme evidenciado pelos resultados fornecidos no estudo, a diabetes pode ser diagnosticada em seus estágios iniciais, analisando variáveis clinicas dos pacientes sem recorrer as obtidas por testes bioquímicos.

Houve outras abordagens para diagnosticar a doença antes, no entanto, elas foram desenvolvidas em ambientes médicos. Isso levantou uma questão se tais doenças podem ser diagnosticadas sem qualquer verificação física da pessoa e sem gastar tempo esperando que a doença agrave. Embora o foco tenha sido estabelecido apenas no diabetes tipo 2, esses métodos também podem ser aplicados a doenças como o câncer, que tem sido uma das principais causas de morte entre as pessoas devido à falta de conhecimento prévio. Isso pode resultar em sérios efeitos, tanto social quanto econômico. Em conclusão, o estudo demonstra a viabilidade do diagnóstico de diabetes sem recorrer a testes bioquímicos.

6.1 Trabalhos Futuros

Esta seção descreve algumas abordagens que podem ser usadas para melhorar a precisão da classificação. Destina-se a ser um guia para futuros pesquisadores que desejam ampliar e desenvolver esta pesquisa. Pode-se destacar como melhorias:

- **Maior amostras de dados:** A primeira e mais óbvia melhoria de todas é utilizar um maior número de dados. Nesta tese, foram consideradas dados de pacientes pertencentes a região nordeste. Uma das tarefas imediatas é aumentar o número de amostras e reavaliar a confiabilidade da estrutura para todo pais.

- **Fatores de controle adicionais:** Os dados podem ser coletados de outras bases, por exemplo, programas de saúdes realizados por organizações não governamentais, de outros países onde brasileiros residem, etc., também para avaliar o padrão. Com atributos adicionais, tais como locais geográficos, etnia, idade, etc., podem ser obtidas informações estatísticas mais finas.

- **Comparação com outros metodos:** Realizar teste de comparação com outra técnicas de diagnósticos de diabetes, tal como o teste de hemoglobina capilar.

REFERÊNCIAS

ABREU, R. V. d. et al. *Avaliação econômica em saúde: desafios para gestão do Sistema Único de Saúde*. [S.l.]: Ministério da Saúde, 2008.

American Diabetes Association et al. Diagnosis and classification of diabetes mellitus. *Diabetes care*, Am Diabetes Assoc, v. 37, n. Supplement 1, p. S81–S90, 2014.

AZEVEDO, A. I. R. L.; SANTOS, M. F. Kdd, semma and crisp-dm: a parallel overview. *IADS-DM*, 2008.

BAHIA, L. R. et al. The costs of type 2 diabetes mellitus outpatient care in the brazilian public health system. *Value in Health*, Elsevier, v. 14, n. 5, p. S137–S140, 2011.

BARBOSA, J. H. P.; OLIVEIRA, S. L. d.; SEARA, L. T. Produtos da glicação avançada dietéticos e as complicações crônicas do diabetes. *Rev. nutr*, v. 22, n. 1, p. 113–124, 2009.

BARCELO, A. et al. The cost of diabetes in latin america and the caribbean. *Bulletin of the world health organization*, SciELO Public Health, v. 81, n. 1, p. 19–27, 2003.

BARTLETT, M. S.; MOVELLAN, J. R.; SEJNOWSKI, T. J. Face recognition by independent component analysis. *Neural Networks, IEEE Transactions on*, IEEE, v. 13, n. 6, p. 1450–1464, 2002.

BORGES, L. C. *Um estudo da Diabetes Mellitus e Hipertensão Arterial baseado em técnicas de Data Mining aplicadas a dados da Administração Regional de Saúde do Centro*. Dissertação (Mestrado) — Instituto Politécnico de Coimbra, 2016.

BORGES, L. C.; MARQUES, V. M.; BERNARDINO, J. Comparison of data mining techniques and tools for data classification. In: ACM. *Proceedings of the International C* Conference on Computer Science and Software Engineering*. [S.l.], 2013. p. 113–116.

BREIMAN, L. Random forests. *Machine learning*, Springer, v. 45, n. 1, p. 5–32, 2001.

BROWN, N. et al. Risk scores based on self-reported or available clinical data to detect undiagnosed type 2 diabetes: a systematic review. *Diabetes research and clinical practice*, Elsevier, v. 98, n. 3, p. 369–385, 2012.

BYEON, B.; RASHEED, K.; DOSHI, P. Enhancing the quality of noisy training data using a genetic algorithm and prototype selection. In: *IC-AI*. [S.l.: s.n.], 2008. p. 821–827.

CANANI, L. H. et al. Prevalência de complicações micro e macrovasculares e de seus fatores de risco em pacientes com diabetes melito do tipo 2 em atendimento ambulatorial. *Rev Assoc Med Bras*, SciELO Brasil, v. 50, n. 3, p. 263–7, 2004.

CASTELLANI, B.; CASTELLANI, J. Data mining: qualitative analysis with health informatics data. *Qualitative Health Research*, Sage Publications, v. 13, n. 7, p. 1005–1018, 2003.

CORREIA, L. O. d. S.; PADILHA, B. M.; VASCONCELOS, S. M. L. Accuracy of registration data for patients with arterial hypertension and diabetes mellitus recorded in the hiperdia system in a state in the northeast of brazil. *Ciência & Saúde Coletiva*, SciELO Public Health, v. 19, n. 6, p. 1685–1697, 2014.

COSTA, D. D.; CAMPOS, L. F.; BARROS, A. K. Classification of breast tissue in mammograms using efficient coding. *Bio-Medical Engineering, On-Line*, v. 10, p. 55, 2011.

DELFOSSE, N.; LOUBATON, P. Adaptive blind separation of independent sources: a deflation approach. *Signal processing*, Elsevier, v. 45, n. 1, p. 59–83, 1995.

DIETTERICH, T. G.; BAKIRI, G. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, v. 2, p. 263–286, 1995.

FERREIRA, L. T. et al. Diabetes melito: hiperglicemia crônica e suas complicações. *Arq bras cienc saúde*, v. 36, n. 3, p. 182–8, 2011.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, Elsevier, v. 55, n. 1, p. 119–139, 1997.

HAMMING, R. W. Error detecting and error correcting codes. *Bell Labs Technical Journal*, Wiley Online Library, v. 29, n. 2, p. 147–160, 1950.

HAN, J.; PEI, J.; KAMBER, M. *Data Mining: Concepts and Techniques*. Elsevier Science, 2011. (The Morgan Kaufmann Series in Data Management Systems). ISBN 9780123814807. Disponível em: <https://books.google.com.br/books?id=pQws07tdpjoC>.

HYVARINEN, A. Fast and robust fixed-point algorithms for independent component analysis. *IEEE transactions on Neural Networks*, IEEE, v. 10, n. 3, p. 626–634, 1999.

HYVÄRINEN, A. The fixed-point algorithm and maximum likelihood estimation for independent component analysis. *Neural Processing Letters*, Springer, v. 10, n. 1, p. 1–5, 1999.

HYVÄRINEN, A.; KARHUNEN, J.; OJA, E. *Independent component analysis*. [S.l.]: John Wiley & Sons, 2004. v. 46.

IDF - Federation Internation Diabetes. Idf diabetes atlas. *Brussels: International Diabetes Federation*, 2015.

III, J. R. G. et al. Report of the expert committee on the diagnosis and classification of diabetes mellitus. *Diabetes care*, American Diabetes Association, v. 20, n. 7, p. 1183, 1997.

INSTITUCIONAIS, I. T. Plano de reorganização da atenção à hipertensão arterial e ao diabetes mellitus. *Rev Saúde Pública*, SciELO Brasil, v. 35, n. 6, p. 585–8, 2001.

KIM, J. et al. Effective representation using ica for face recognition robust to local distortion and partial occlusion. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, IEEE, v. 27, n. 12, p. 1977–1981, 2005.

LEE, C.-S.; WANG, M.-H. A fuzzy expert system for diabetes decision support application. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, IEEE, v. 41, n. 1, p. 139–153, 2011.

LUCENA, F. et al. Statistical coding and decoding of heartbeat intervals. *PloS one*, Public Library of Science, v. 6, n. 6, p. e20227, 2011.

MALTA, D. C.; MERHY, E. E. O percurso da linha do cuidado sob a perspectiva das doenças crônicas não transmissíveis. *Interface (Botucatu)*, SciELO Brasil, v. 14, n. 34, p. 593–605, 2010.

MARINHO, M. G. da S. et al. Análise de custos da assistência à saúde aos portadores de diabetes melito e hipertensão arterial em uma unidade de saúde pública de referência em recife–brasil. *Arq bras endocrinol metab*, v. 55, n. 6, p. 406, 2011.

MEYER, D.; LEISCH, F.; HORNIK, K. The support vector machine under test. *Neurocomputing*, Elsevier, v. 55, n. 1, p. 169–186, 2003.

MILECH, A. et al. Diretrizes da sociedade brasileira de diabetes (2015-2016). *A.C. Farmacêutica*, 2016.

OJA, E.; YUAN, Z. The fastica algorithm revisited: Convergence analysis. *IEEE Transactions on Neural Networks*, IEEE, v. 17, n. 6, p. 1370–1381, 2006.

OTERO, F. E.; FREITAS, A. A.; JOHNSON, C. G. Inducing decision trees with an ant colony optimization algorithm. *Applied Soft Computing*, Elsevier, v. 12, n. 11, p. 3615–3626, 2012.

PATIL, B. M.; JOSHI, R. C.; TOSHNIWAL, D. Hybrid prediction model for type-2 diabetic patients. *Expert systems with applications*, Elsevier, v. 37, n. 12, p. 8102–8108, 2010.

QUINLAN, J. *C4.5: Programs for Machine Learning*. Elsevier Science, 2014. (Ebrary online). ISBN 9780080500584. Disponível em: <https://books.google.com.br/books?id=b3ujBQAAQBAJ>.

RIBEIRO, Á. C. et al. Diabetes classification using a redundancy reduction preprocessor. *Research on Biomedical Engineering*, SciELO Brasil, v. 31, n. 2, p. 97–106, 2015.

RIBEIRO, A. C. et al. Tracking type 2 diabetes using sparse coding. In: AMER DIABETES ASSOC 1701 N BEAUREGARD ST, ALEXANDRIA, VA 22311-1717 USA. *Diabetes*. [S.l.], 2015. v. 64, p. A409–A409.

SARAIVA, J. F. K. et al. Diabetes mellitus no brasil: características clínicas, padrão de tratamento e custos associados ao cuidado da doença. *J. bras. econ. saúde (Impr.)*, v. 8, n. 2, p. 80–90, 2016.

WEI, T. A study of the fixed points and spurious solutions of the deflation-based fastica algorithm. *Neural Computing and Applications*, Springer, v. 28, n. 1, p. 13–24, 2017.

YASASWI, B.; PRAJNA, B. The early augmentationfor diabetes diagnosis using data mining approaches. *International Journal of Computer Science And Technology*, v. 7, n. 3, p. 27–31, 2016.

YOO, I. et al. Data mining in healthcare and biomedicine: a survey of the literature. *Journal of medical systems*, Springer, v. 36, n. 4, p. 2431–2448, 2012.

APÊNDICE A – CÓDIGO MATLAB: NORDESTEBRAZIL.M

```
clear; close all;clc

% Carregar bibliotecas
addpath('../../libs/FastICA')
addpath('../../libs/FastICA/CVS')
addpath('./classificadores')

% inicializacao das variaveis
basePath='../../base/';
regiao='Nordeste';
DiabeticosRegiao={['Diabeticos-' regiao]};
naoDiabeticosRegiao={['Hipertensos-' regiao]};
nBins=100;
i=1;

% Carregar a base de dados;
load([basePath DiabeticosRegiao{i}]);
load([basePath naoDiabeticosRegiao{i}]);

% Remove o hifem do nome da variavel
diabeticos=eval(DiabeticosRegiao{i}([1:10 12:end]));
naoDiabeticos=eval(naoDiabeticosRegiao{i}([1:11 13:end]));

% Ordenar os elementos
diabeticos=sortrows(diabeticos,'GLICEMIA_EM_JEJUM','ascend');
naoDiabeticos=sortrows(naoDiabeticos,'GLICEMIA_EM_JEJUM','ascend
    ');

% Selecao das variaveis clinicas
diabeticos=diabeticos(:,[2:8 10:14]);
diabeticosLabel=ones(size(diabeticos,1),1);

naoDiabeticos=naoDiabeticos(:,[2:8 10:14]);
naoDiabeticosLabel=zeros(size(naoDiabeticos,1),1);

MDAD = table2array(varfun(@double, diabeticos));
```

```
36 MDAND = table2array(varfun(@double, naoDiabeticos));
37
38 % valores 'inf' serao tratados como faltando para na PCA.
39 MDAD(isinf(MDAD)) = NaN;
40 MDAND(isinf(MDAND)) = NaN;
41
42 numericDataset=[MDAD ; MDAND];
43 numericDatasetLabel=[diabeticosLabel; naoDiabeticosLabel];
44
45 MNR=performICAonDataset(numericDataset,MDAD,MDAND);
46
47 % Normalizacao da matriz MNR
48 MNRN = bsxfun(@rdivide,bsxfun(@minus,MNR ,min(MNR)),max(MNR)-min
       (MNR));
49
50 % Matriz para classificador
51 MPC=[MNRN numericDatasetLabel];
52
53 %% Testar os diferentes tipos de classificadores
54
55 % Treinando os classificadores do tipo arvores.
56 [trainedClassifierTree, validationAccuracyTree, confusionMatTree
       ] = trainClassifierTree(MPC);
57
58 % Treinando os classificadores do tipo SVM.
59 [trainedClassifierSVM, validationAccuracySVM, confusionMatSVM] =
        trainClassifierSVM(MPC);
60
61 % Treinando os classificadores do tipo KNN.
62 [trainedClassifierKNN, validationAccuracyKNN, confusionMatKNN] =
        trainClassifierKNN(MPC);
63
64 % Treinando os classificadores do tipo Ensemble.
65 [trainedClassifierEnsemble, validationAccuracyEnsemble,
       confusionMatEnsemble] = trainClassifierEnsemble(MPC);
66
67 %% Resultados
68
```

```
69 resultadosTree={'Tree-ComplexTree',validationAccuracyTree.
       complex,confusionMatTree.complex;...
70       'Tree-MediumTree',validationAccuracyTree.medium,
             confusionMatTree.medium; ...
71       'Tree-SimpleTree',validationAccuracyTree.simple,
             confusionMatTree.simple };
72
73 resultadosSVM ={'SVM-Linear',validationAccuracySVM.linear,
       confusionMatSVM.linear; ...
74       'SVM-Quadratic',validationAccuracySVM.quadratic,
             confusionMatSVM.quadratic; ...
75       'SVM-Cubic',validationAccuracySVM.cubic,confusionMatSVM.
             cubic; ...
76       'SVM-FineGaussian',validationAccuracySVM.fineGaussian,
             confusionMatSVM.fineGaussian; ...
77       'SVM-MediumGaussian',validationAccuracySVM.mediumGaussian,
             confusionMatSVM.mediumGaussian; ...
78       'SVM-CoarseGaussian',validationAccuracySVM.coarseGaussian,
             confusionMatSVM.coarseGaussian};
79
80 resultadosKNN = {'KNN-Fine', validationAccuracyKNN.fine,
       confusionMatKNN.fine; ...
81       'KNN-Medium',validationAccuracyKNN.medium,confusionMatKNN.
             medium; ...
82       'KNN-Coarse',validationAccuracyKNN.coarse,confusionMatKNN.
             coarse; ...
83       'KNN-Cosine',validationAccuracyKNN.cosine,confusionMatKNN.
             cosine; ...
84       'KNN-Cubic',validationAccuracyKNN.cubic,confusionMatKNN.
             cubic; ...
85       'KNN-Weighted',validationAccuracyKNN.weighted,
             confusionMatKNN.weighted};
86
87 resultadosEnsemble={'Ensemble-BoostedTrees',
       validationAccuracyEnsemble.boostedTrees,confusionMatEnsemble.
       boostedTrees; ...
88       'Ensemble-BaggedTrees', validationAccuracyEnsemble.
             baggedTrees,confusionMatEnsemble.baggedTrees; ...
```

```
89      'Ensemble-SubspaceDiscriminant',validationAccuracyEnsemble.
            subDiscriminant,confusionMatEnsemble.subDiscriminant; ...
90      'Ensemble-SubspaceKNN', validationAccuracyEnsemble.subKNN,
            confusionMatEnsemble.subKNN; ...
91      'Ensemble-RUSBoostedTrees',validationAccuracyEnsemble.
            rusBoostedTrees,confusionMatEnsemble.rusBoostedTrees};
92
93  resultados={'Tree',resultadosTree;'SVM',resultadosSVM;...
94      'KNN',resultadosKNN;'Ensemble',resultadosEnsemble;};
95
96  resultadosTodos={resultadosTree{:,1}, resultadosSVM{:,1},...
97      resultadosKNN{:,1}, resultadosEnsemble{:,1};...
98      resultadosTree{:,2}, resultadosSVM{:,2},...
99      resultadosKNN{:,2}, resultadosEnsemble{:,2};...
100     resultadosTree{:,3}, resultadosSVM{:,3},...
101     resultadosKNN{:,3}, resultadosEnsemble{:,3}}';
102
103 resultadosTodosOrdenado = flipud(sortrows(resultadosTodos,2));
```

APÊNDICE B – CÓDIGO MATLAB USA PARA OS CLASSIFICADORES DO TIPO ÁRVORE

```
function [trainedClassifier, validationAccuracy, confusionMat] = trainClassifierTree(trainingData)

% Extract predictors and response
% This code processes the data into the right shape for training the
% classifier.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_7', 'column_8', 'column_9', 'column_10', 'column_11', 'column_12', 'column_13'});

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_7', 'column_8', 'column_9', 'column_10', 'column_11', 'column_12'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_13;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, false, false, false];

% Train a classifier
% Especificacao da configuracao do classificador arvore simples
classificationTreeSimple = fitctree(predictors, response, ...
    'SplitCriterion', 'gdi', 'MaxNumSplits', 4, ...
    'ClassNames', [0; 1]);

% Especificacao da configuracao do classificador arvore media
classificationTreeMedium = fitctree(predictors, response, ...
    'SplitCriterion', 'gdi', 'MaxNumSplits', 20, ...
    'ClassNames', [0; 1]);

% Especificacao da configuracao do classificador arvore complexa
classificationTreeComplex = fitctree(predictors, response, ...
```

```
27      'SplitCriterion', 'gdi', 'MaxNumSplits', 100, ...
28      'ClassNames', [0; 1]);
29
30 % Create the result struct with predict function
31 predictorExtractionFcn = @(x) array2table(x, 'VariableNames',
       predictorNames);
32 %treePredictFcn = @(x) predict(classificationTree, x);
33 simpleTreePredictFcn = @(x) predict(classificationTreeSimple, x)
       ;
34 mediumTreePredictFcn = @(x) predict(classificationTreeMedium, x)
       ;
35 complexTreePredictFcn = @(x) predict(classificationTreeComplex,
       x);
36 %trainedClassifier.predictFcn = @(x) treePredictFcn(
       predictorExtractionFcn(x));
37 trainedClassifier.simplePredictFcn = @(x) simpleTreePredictFcn(
       predictorExtractionFcn(x));
38 trainedClassifier.mediumPredictFcn = @(x) mediumTreePredictFcn(
       predictorExtractionFcn(x));
39 trainedClassifier.complexPredictFcn = @(x) complexTreePredictFcn
       (predictorExtractionFcn(x));
40
41 % Add additional fields to the result struct
42 %trainedClassifier.ClassificationTree = classificationTree;
43 trainedClassifier.ClassificationTreeSimple =
       classificationTreeSimple;
44 trainedClassifier.ClassificationTreeMedium =
       classificationTreeMedium;
45 trainedClassifier.ClassificationTreeComplex =
       classificationTreeComplex;
46 trainedClassifier.About = 'This struct is a trained classifier
       exported from Classification Learner R2016a.';
47 trainedClassifier.HowToPredict = sprintf('To make predictions on
        a new predictor column matrix, X, use: \n  yfit = c.
       predictFcn(X) \nreplacing ''c'' with the name of the variable
        that is this struct, e.g. ''trainedClassifier''. \n \nX must
        contain exactly 12 columns because this classifier was
       trained using 12 predictors. \nX must contain only predictor
       columns in exactly the same order and format as your training
```

```
    \ndata. Do not include the response column or any columns
    you did not import into \nClassification Learner. \n \nFor
    more information, see <a href="matlab:helpview(fullfile(
    docroot, ''stats'', ''stats.map''), ''
    appclassification_exportmodeltoworkspace'')">How to predict
    using an exported model</a>.');
48
49 % Extract predictors and response
50 % This code processes the data into the right shape for training
     the
51 % classifier.
52 % Convert input to table
53 inputTable = array2table(trainingData, 'VariableNames', {'
    column_1', 'column_2', 'column_3', 'column_4', 'column_5', '
    column_6', 'column_7', 'column_8', 'column_9', 'column_10', '
    column_11', 'column_12', 'column_13'});
54
55 predictorNames = {'column_1', 'column_2', 'column_3', 'column_4'
    , 'column_5', 'column_6', 'column_7', 'column_8', 'column_9',
     'column_10', 'column_11', 'column_12'};
56 predictors = inputTable(:, predictorNames);
57 response = inputTable.column_13;
58 isCategoricalPredictor = [false, false, false, false, false,
    false, false, false, false, false, false, false];
59
60 % Perform cross-validation
61 %partitionedModel = crossval(trainedClassifier.
    ClassificationTree, 'KFold', 20);
62 partitionedModel.simple = crossval(trainedClassifier.
    ClassificationTreeSimple, 'KFold', 20);
63 partitionedModel.medium = crossval(trainedClassifier.
    ClassificationTreeMedium, 'KFold', 20);
64 partitionedModel.complex = crossval(trainedClassifier.
    ClassificationTreeComplex, 'KFold', 20);
65
66 % Compute validation accuracy
67 %validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun',
     'ClassifError');
```

```
68 validationAccuracy.simple = 1 - kfoldLoss(partitionedModel.
       simple, 'LossFun', 'ClassifError');
69 validationAccuracy.medium = 1 - kfoldLoss(partitionedModel.
       medium, 'LossFun', 'ClassifError');
70 validationAccuracy.complex = 1 - kfoldLoss(partitionedModel.
       complex, 'LossFun', 'ClassifError');
71
72 % Compute validation predictions and scores
73 [validationPredictions.simple, validationScores.simple] =
       kfoldPredict(partitionedModel.simple);
74 [validationPredictions.medium, validationScores.medium] =
       kfoldPredict(partitionedModel.medium);
75 [validationPredictions.complex, validationScores.complex] =
       kfoldPredict(partitionedModel.complex);
76 confusionMat.simple=confusionmat(response,validationPredictions.
       simple,'order',[1 0]);
77 confusionMat.medium=confusionmat(response,validationPredictions.
       medium,'order',[1 0]);
78 confusionMat.complex=confusionmat(response,validationPredictions
       .complex,'order',[1 0]);
```

APÊNDICE C – CÓDIGO MATLAB USA PARA OS CLASSIFICADORES DO TIPO SVM

```
function [trainedClassifier, validationAccuracy, confusionMat] = trainClassifierSVM(trainingData)

% Extract predictors and response
% This code processes the data into the right shape for training the
% classifier.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_7', 'column_8', 'column_9', 'column_10', 'column_11', 'column_12', 'column_13'});

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_7', 'column_8', 'column_9', 'column_10', 'column_11', 'column_12'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_13;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, false, false, false];

% Train a classifier
% Esse trecho especifica a configuracao do classificador svm linear
classificationLinearSVM = fitcsvm(predictors, response, ...
    'KernelFunction', 'linear', 'PolynomialOrder', [], ...
    'KernelScale', 'auto', 'Standardize', true, ...
    'ClassNames', [0; 1]);

% Esse trecho especifica a configuracao do classificador svm quadratica
classificationQuadraticSVM = fitcsvm(predictors, response, ...
    'KernelFunction', 'polynomial', 'PolynomialOrder', 2, ...
    'KernelScale', 'auto', 'Standardize', true, ...
```

```
25      'ClassNames', [0; 1]);
26
27  % Esse trecho especifica a configuracao do classificador svm
        cubica
28  classificationCubicSVM = fitcsvm(predictors, response, ...
29      'KernelFunction', 'polynomial', 'PolynomialOrder', 3, ...
30      'KernelScale', 'auto', 'Standardize', true,  ...
31      'ClassNames', [0; 1]);
32
33  % Esse trecho especifica a configuracao do classificador svm
        gaussiana fina
34  classificationFineGaussianSVM = fitcsvm(predictors, response,
        ...
35      'KernelFunction', 'gaussian', 'PolynomialOrder', [], ...
36      'KernelScale', 0.87, 'Standardize', true,  ...
37      'ClassNames', [0; 1]);
38
39  % Esse trecho especifica a configuracao do classificador svm
        gaussiana media
40  classificationMediumGaussianSVM = fitcsvm(predictors, response,
        ...
41      'KernelFunction', 'gaussian', 'PolynomialOrder', [], ...
42      'KernelScale', 3.5, 'Standardize', true,  ...
43      'ClassNames', [0; 1]);
44
45  % Esse trecho especifica a configuracao do classificador svm
        gaussiana larga
46  classificationCoarseGaussianSVM = fitcsvm(predictors, response,
        ...
47      'KernelFunction', 'gaussian', 'PolynomialOrder', [], ...
48      'KernelScale', 14, 'Standardize', true,  ...
49      'ClassNames', [0; 1]);
50
51
52  % Create the result struct with predict function
53  predictorExtractionFcn = @(x) array2table(x, 'VariableNames',
        predictorNames);
54
55
```

```
56 %svmPredictFcn = @(x) predict(classificationSVM, x);
57 lienarSvmPredictFcn = @(x) predict(classificationLinearSVM, x);
58 quadraticSvmPredictFcn = @(x) predict(classificationQuadraticSVM
       , x);
59 cubicSvmPredictFcn = @(x) predict(classificationCubicSVM, x);
60 fineGaussianSvmPredictFcn = @(x) predict(
       classificationFineGaussianSVM, x);
61 mediumGaussianSvmPredictFcn = @(x) predict(
       classificationMediumGaussianSVM, x);
62 coarseGaussianSvmPredictFcn = @(x) predict(
       classificationCoarseGaussianSVM, x);
63
64 trainedClassifier.linearPredictFcn = @(x) lienarSvmPredictFcn(
       predictorExtractionFcn(x));
65 trainedClassifier.quadraticPredictFcn = @(x)
       quadraticSvmPredictFcn(predictorExtractionFcn(x));
66 trainedClassifier.cubicPredictFcn = @(x) cubicSvmPredictFcn(
       predictorExtractionFcn(x));
67 trainedClassifier.fineGaussianPredictFcn = @(x)
       fineGaussianSvmPredictFcn(predictorExtractionFcn(x));
68 trainedClassifier.mediumGaussianPredictFcn = @(x)
       mediumGaussianSvmPredictFcn(predictorExtractionFcn(x));
69 trainedClassifier.coarseGaussianPredictFcn = @(x)
       coarseGaussianSvmPredictFcn(predictorExtractionFcn(x));
70
71 % Add additional fields to the result struct
72 trainedClassifier.ClassificationLinearSVM =
       classificationLinearSVM;
73 trainedClassifier.ClassificationQuadraticSVM =
       classificationQuadraticSVM;
74 trainedClassifier.ClassificationCubicSVM =
       classificationCubicSVM;
75 trainedClassifier.ClassificationFineGaussianSVM =
       classificationFineGaussianSVM;
76 trainedClassifier.ClassificationMediumGaussianSVM =
       classificationMediumGaussianSVM;
77 trainedClassifier.ClassificationCoarseGaussianSVM =
       classificationCoarseGaussianSVM;
78
```

```
79 trainedClassifier.About = 'This struct is a trained classifier
   exported from Classification Learner R2016a.';
80 trainedClassifier.HowToPredict = sprintf('To make predictions on
    a new predictor column matrix, X, use: \n  yfit = c.
   predictFcn(X) \nreplacing ''c'' with the name of the variable
    that is this struct, e.g. ''trainedClassifier''. \n \nX must
    contain exactly 12 columns because this classifier was
   trained using 12 predictors. \nX must contain only predictor
   columns in exactly the same order and format as your training
    \ndata. Do not include the response column or any columns
   you did not import into \nClassification Learner. \n \nFor
   more information, see <a href="matlab:helpview(fullfile(
   docroot, ''stats'', ''stats.map''), ''
   appclassification_exportmodeltoworkspace'')">How to predict
   using an exported model</a>.');
81
82 % Extract predictors and response
83 % This code processes the data into the right shape for training
    the
84 % classifier.
85 % Convert input to table
86 inputTable = array2table(trainingData, 'VariableNames', {'
   column_1', 'column_2', 'column_3', 'column_4', 'column_5', '
   column_6', 'column_7', 'column_8', 'column_9', 'column_10', '
   column_11', 'column_12', 'column_13'});
87
88 predictorNames = {'column_1', 'column_2', 'column_3', 'column_4'
   , 'column_5', 'column_6', 'column_7', 'column_8', 'column_9',
    'column_10', 'column_11', 'column_12'};
89 predictors = inputTable(:, predictorNames);
90 response = inputTable.column_13;
91 isCategoricalPredictor = [false, false, false, false, false,
   false, false, false, false, false, false, false];
92
93 % Perform cross-validation
94 partitionedModel.linear = crossval(trainedClassifier.
   ClassificationLinearSVM, 'KFold', 20);
95 partitionedModel.quadratic = crossval(trainedClassifier.
   ClassificationQuadraticSVM, 'KFold', 20);
```

```
96  partitionedModel.cubic = crossval(trainedClassifier.
        ClassificationCubicSVM, 'KFold', 20);
97  partitionedModel.fineGaussian = crossval(trainedClassifier.
        ClassificationFineGaussianSVM, 'KFold', 20);
98  partitionedModel.mediumGaussian = crossval(trainedClassifier.
        ClassificationMediumGaussianSVM, 'KFold', 20);
99  partitionedModel.coarseGaussian = crossval(trainedClassifier.
        ClassificationCoarseGaussianSVM, 'KFold', 20);
100
101 % Compute validation accuracy
102 validationAccuracy.linear = 1 - kfoldLoss(partitionedModel.
        linear, 'LossFun', 'ClassifError');
103 validationAccuracy.quadratic = 1 - kfoldLoss(partitionedModel.
        quadratic, 'LossFun', 'ClassifError');
104 validationAccuracy.cubic = 1 - kfoldLoss(partitionedModel.cubic,
         'LossFun', 'ClassifError');
105 validationAccuracy.fineGaussian = 1 - kfoldLoss(partitionedModel
        .fineGaussian, 'LossFun', 'ClassifError');
106 validationAccuracy.mediumGaussian = 1 - kfoldLoss(
        partitionedModel.mediumGaussian, 'LossFun', 'ClassifError');
107 validationAccuracy.coarseGaussian = 1 - kfoldLoss(
        partitionedModel.coarseGaussian, 'LossFun', 'ClassifError');
108
109
110 % Compute validation predictions and scores
111 [validationPredictions.linear, validationScores.linear] =
        kfoldPredict(partitionedModel.linear);
112 [validationPredictions.quadratic, validationScores.quadratic] =
        kfoldPredict(partitionedModel.quadratic);
113 [validationPredictions.cubic, validationScores.cubic] =
        kfoldPredict(partitionedModel.cubic);
114 [validationPredictions.fineGaussian, validationScores.
        fineGaussian] = kfoldPredict(partitionedModel.fineGaussian);
115 [validationPredictions.mediumGaussian, validationScores.
        mediumGaussian] = kfoldPredict(partitionedModel.
        mediumGaussian);
116 [validationPredictions.coarseGaussian, validationScores.
        coarseGaussian] = kfoldPredict(partitionedModel.
        coarseGaussian);
```

```
117
118 confusionMat.linear=confusionmat(response,validationPredictions.
        linear,'order',[1 0]);
119 confusionMat.quadratic=confusionmat(response,
        validationPredictions.quadratic,'order',[1 0]);
120 confusionMat.cubic=confusionmat(response,validationPredictions.
        cubic,'order',[1 0]);
121 confusionMat.fineGaussian=confusionmat(response,
        validationPredictions.fineGaussian,'order',[1 0]);
122 confusionMat.mediumGaussian=confusionmat(response,
        validationPredictions.mediumGaussian,'order',[1 0]);
123 confusionMat.coarseGaussian=confusionmat(response,
        validationPredictions.coarseGaussian,'order',[1 0]);
```

APÊNDICE D – CÓDIGO MATLAB USA PARA OS CLASSIFICADORES DO TIPO KNN

```
function [trainedClassifier, validationAccuracy, confusionMat] = trainClassifierKNNFineKNN(trainingData)

% Extract predictors and response
% This code processes the data into the right shape for training the
% classifier.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_7', 'column_8', 'column_9', 'column_10', 'column_11', 'column_12', 'column_13'});

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_7', 'column_8', 'column_9', 'column_10', 'column_11', 'column_12'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_13;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, false, false, false];

% Train a classifier
% Esse trecho especifica a configuracao do classificador KNN fino
classificationKNNFine = fitcknn(predictors, response, ...
    'Distance', 'Euclidean', 'Exponent', [], ...
    'NumNeighbors', 1, 'DistanceWeight', 'Equal', ...
    'Standardize', true, 'ClassNames', [0; 1]);

% Esse trecho especifica a configuracao do classificador KNN medio
classificationKNNMedium = fitcknn(predictors, response, ...
    'Distance', 'Euclidean', 'Exponent', [], ...
    'NumNeighbors', 10, 'DistanceWeight', 'Equal', ...
```

```
25      'Standardize', true, 'ClassNames', [0; 1]);
26
27  % Esse trecho especifica a configuracao do classificador KNN
        largo
28  classificationKNNCoarse = fitcknn(predictors, response, ...
29      'Distance', 'Euclidean', 'Exponent', [], ...
30      'NumNeighbors', 100, 'DistanceWeight', 'Equal', ...
31      'Standardize', true, 'ClassNames', [0; 1]);
32
33  % Esse trecho especifica a configuracao do classificador KNN
        cubico
34  classificationKNNCubic = fitcknn(predictors, response, ...
35      'Distance', 'Minkowski', 'Exponent', 3, ...
36      'NumNeighbors', 10, 'DistanceWeight', 'Equal', ...
37      'Standardize', true, 'ClassNames', [0; 1]);
38
39  % Esse trecho especifica a configuracao do classificador KNN
        coseno
40  classificationKNNCosine = fitcknn(predictors, response, ...
41      'Distance', 'Cosine', 'Exponent', [], ...
42      'NumNeighbors', 10, 'DistanceWeight', 'Equal', ...
43      'Standardize', true, 'ClassNames', [0; 1]);
44
45  % Esse trecho especifica a configuracao do classificador KNN
        ponderado
46  classificationKNNWeighted = fitcknn(predictors, response, ...
47      'Distance', 'Euclidean', 'Exponent', [], ...
48      'NumNeighbors', 10, 'DistanceWeight', 'SquaredInverse', ...
49      'Standardize', true, 'ClassNames', [0; 1]);
50
51  % Create the result struct with predict function
52  predictorExtractionFcn = @(x) array2table(x, 'VariableNames',
        predictorNames);
53
54  fineKNNPredictFcn = @(x) predict(classificationKNNFine, x);
55  mediumKNNPredictFcn = @(x) predict(classificationKNNMedium, x);
56  coarseKNNPredictFcn = @(x) predict(classificationKNNCoarse, x);
57  cubicKNNPredictFcn = @(x) predict(classificationKNNCubic, x);
58  cosineKNNPredictFcn = @(x) predict(classificationKNNCosine, x);
```

```
59 weighthedKNNPredictFcn = @(x) predict(classificationKNNWeighted,
       x);
60
61
62 trainedClassifier.fineKNNPredictFcn = @(x) fineKNNPredictFcn(
       predictorExtractionFcn(x));
63 trainedClassifier.mediumKNNPredictFcn = @(x) mediumKNNPredictFcn
       (predictorExtractionFcn(x));
64 trainedClassifier.coarseKNNPredictFcn = @(x) coarseKNNPredictFcn
       (predictorExtractionFcn(x));
65 trainedClassifier.cubicKNNPredictFcn = @(x) cubicKNNPredictFcn(
       predictorExtractionFcn(x));
66 trainedClassifier.cosineKNNPredictFcn = @(x) cosineKNNPredictFcn
       (predictorExtractionFcn(x));
67 trainedClassifier.weighthedKNNPredictFcn = @(x)
       weighthedKNNPredictFcn(predictorExtractionFcn(x));
68
69 % Add additional fields to the result struct
70 trainedClassifier.classificationKNNFine = classificationKNNFine;
71 trainedClassifier.classificationKNNMedium =
       classificationKNNMedium;
72 trainedClassifier.classificationKNNCoarse =
       classificationKNNCoarse;
73 trainedClassifier.classificationKNNCubic =
       classificationKNNCubic;
74 trainedClassifier.classificationKNNCosine =
       classificationKNNCosine;
75 trainedClassifier.classificationKNNWeighted =
       classificationKNNWeighted;
76
77 trainedClassifier.About = 'This struct is a trained classifier
       exported from Classification Learner R2016a.';
78 trainedClassifier.HowToPredict = sprintf('To make predictions on
       a new predictor column matrix, X, use: \n  yfit = c.
       predictFcn(X) \nreplacing ''c'' with the name of the variable
       that is this struct, e.g. ''trainedClassifier''. \n \nX must
       contain exactly 12 columns because this classifier was
       trained using 12 predictors. \nX must contain only predictor
       columns in exactly the same order and format as your training
```

```
    \ndata. Do not include the response column or any columns
    you did not import into \nClassification Learner. \n \nFor
    more information, see <a href="matlab:helpview(fullfile(
    docroot, ''stats'', ''stats.map''), ''
    appclassification_exportmodeltoworkspace'')">How to predict
    using an exported model</a>.');
79
80 % Extract predictors and response
81 % This code processes the data into the right shape for training
     the
82 % classifier.
83 % Convert input to table
84 inputTable = array2table(trainingData, 'VariableNames', {'
    column_1', 'column_2', 'column_3', 'column_4', 'column_5', '
    column_6', 'column_7', 'column_8', 'column_9', 'column_10', '
    column_11', 'column_12', 'column_13'});
85
86 predictorNames = {'column_1', 'column_2', 'column_3', 'column_4'
    , 'column_5', 'column_6', 'column_7', 'column_8', 'column_9',
     'column_10', 'column_11', 'column_12'};
87 predictors = inputTable(:, predictorNames);
88 response = inputTable.column_13;
89 isCategoricalPredictor = [false, false, false, false, false,
    false, false, false, false, false, false, false];
90
91 % Perform cross-validation
92 partitionedModel.fine = crossval(trainedClassifier.
    classificationKNNFine, 'KFold', 20);
93 partitionedModel.medium = crossval(trainedClassifier.
    classificationKNNMedium, 'KFold', 20);
94 partitionedModel.coarse = crossval(trainedClassifier.
    classificationKNNCoarse, 'KFold', 20);
95 partitionedModel.cubic = crossval(trainedClassifier.
    classificationKNNCubic, 'KFold', 20);
96 partitionedModel.cosine = crossval(trainedClassifier.
    classificationKNNCosine, 'KFold', 20);
97 partitionedModel.weighted = crossval(trainedClassifier.
    classificationKNNWeighted, 'KFold', 20);
98
```

```
99 % Compute validation accuracy
100 validationAccuracy.fine = 1 - kfoldLoss(partitionedModel.fine, '
    LossFun', 'ClassifError');
101 validationAccuracy.medium = 1 - kfoldLoss(partitionedModel.
    medium, 'LossFun', 'ClassifError');
102 validationAccuracy.coarse = 1 - kfoldLoss(partitionedModel.
    coarse, 'LossFun', 'ClassifError');
103 validationAccuracy.cubic = 1 - kfoldLoss(partitionedModel.cubic,
     'LossFun', 'ClassifError');
104 validationAccuracy.cosine = 1 - kfoldLoss(partitionedModel.
    cosine, 'LossFun', 'ClassifError');
105 validationAccuracy.weighted = 1 - kfoldLoss(partitionedModel.
    weighted, 'LossFun', 'ClassifError');
106
107 % Compute validation predictions and scores
108 [validationPredictions.fine, validationScores] = kfoldPredict(
    partitionedModel.fine);
109 [validationPredictions.medium, validationScores] = kfoldPredict(
    partitionedModel.medium);
110 [validationPredictions.coarse, validationScores] = kfoldPredict(
    partitionedModel.coarse);
111 [validationPredictions.cubic, validationScores] = kfoldPredict(
    partitionedModel.cubic);
112 [validationPredictions.cosine, validationScores] = kfoldPredict(
    partitionedModel.cosine);
113 [validationPredictions.weighted, validationScores] =
    kfoldPredict(partitionedModel.weighted);
114
115 confusionMat.fine=confusionmat(response,validationPredictions.
    fine,'order',[1 0]);
116 confusionMat.medium=confusionmat(response,validationPredictions.
    medium,'order',[1 0]);
117 confusionMat.coarse=confusionmat(response,validationPredictions.
    coarse,'order',[1 0]);
118 confusionMat.cubic=confusionmat(response,validationPredictions.
    cubic,'order',[1 0]);
119 confusionMat.cosine=confusionmat(response,validationPredictions.
    cosine,'order',[1 0]);
```

```
120 confusionMat.weighted=confusionmat(response,
        validationPredictions.weighted,'order',[1  0]);
```

APÊNDICE E – CÓDIGO MATLAB USA PARA OS CLASSIFICADORES DO TIPO ENSEMBLE

```
function [trainedClassifier, validationAccuracy, confusionMat] = trainClassifierEnsembleBaggedTrees(trainingData)

% Extract predictors and response
% This code processes the data into the right shape for training the
% classifier.
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_7', 'column_8', 'column_9', 'column_10', 'column_11', 'column_12', 'column_13'});

predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_7', 'column_8', 'column_9', 'column_10', 'column_11', 'column_12'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_13;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, false, false, false];

% Train a classifier
% Esse trecho especifica a configuracao do classificador ensemble de arvores bagged
classificationEnsembleBaggedTrees = fitensemble(predictors, response, ...
    'Bag', 30, 'Tree', ...
    'Type', 'Classification', 'ClassNames', [0; 1]);

% Esse trecho especifica a configuracao do classificador ensemble de arvores boosted
template = templateTree('MaxNumSplits', 20);
classificationEnsembleBoostedTrees = fitensemble(predictors, response, ...
```

```
23      'AdaBoostM1', 30, template, ...
24      'Type', 'Classification', ...
25      'LearnRate', 0.1, 'ClassNames', [0; 1]);
26
27  % Esse trecho especifica a configuracao do classificador
        ensemble de arvores RUSBoosted
28  classificationEnsembleRUSBoostedTrees = fitensemble(predictors,
        response, ...
29      'RUSBoost', 30, template, ...
30      'Type', 'Classification', ...
31      'LearnRate', 0.1, 'ClassNames', [0; 1]);
32
33  % Esse trecho especifica a configuracao do classificador
        ensemble com subespaco discriminante
34  subspaceDimension = max(1, min(6, width(predictors) - 1));
35  classificationEnsembleSubDiscriminant = fitensemble(predictors,
        response, ...
36      'Subspace', 30, 'Discriminant', ...
37      'Type', 'Classification', ...
38      'NPredToSample', subspaceDimension, 'ClassNames', [0; 1]);
39
40  % Esse trecho especifica a configuracao do classificador
        ensemble com subespaco KNN
41  template = templateTree('MaxNumSplits', 20);
42  subspaceDimension = max(1, min(6, width(predictors) - 1));
43  classificationEnsembleSubKNN = fitensemble(predictors, response,
        ...
44      'Subspace', 30, 'KNN', ...
45      'Type', 'Classification', ...
46      'NPredToSample', subspaceDimension, 'ClassNames', [0; 1]);
47
48
49
50
51  % Create the result struct with predict function
52  predictorExtractionFcn = @(x) array2table(x, 'VariableNames',
        predictorNames);
53
54
```

```
55 %ensemblePredictFcn = @(x) predict(classificationEnsemble, x);
56 baggedTreesEnsemblePredictFcn = @(x) predict(
       classificationEnsembleBaggedTrees, x);
57 boostedTreesEnsemblePredictFcn = @(x) predict(
       classificationEnsembleBoostedTrees, x);
58 rusBoostedTreesEnsemblePredictFcn = @(x) predict(
       classificationEnsembleRUSBoostedTrees, x);
59 subDiscriminantEnsemblePredictFcn = @(x) predict(
       classificationEnsembleSubDiscriminant, x);
60 subKNNEnsemblePredictFcn = @(x) predict(
       classificationEnsembleSubKNN, x);
61
62 trainedClassifier.baggedTreesPredictFcn = @(x)
       baggedTreesEnsemblePredictFcn(predictorExtractionFcn(x));
63 trainedClassifier.boostedTreesPredictFcn = @(x)
       boostedTreesEnsemblePredictFcn(predictorExtractionFcn(x));
64 trainedClassifier.rusBoostedTreesPredictFcn = @(x)
       rusBoostedTreesEnsemblePredictFcn(predictorExtractionFcn(x));
65 trainedClassifier.subDiscriminantPredictFcn = @(x)
       subDiscriminantEnsemblePredictFcn(predictorExtractionFcn(x));
66 trainedClassifier.subKNNEnsemblePredictFcn = @(x)
       subKNNEnsemblePredictFcn(predictorExtractionFcn(x));
67
68 % Add additional fields to the result struct
69 %trainedClassifier.ClassificationEnsemble =
       classificationEnsemble;
70 trainedClassifier.ClassificationEnsembleBaggedTrees =
       classificationEnsembleBaggedTrees;
71 trainedClassifier.ClassificationEnsembleBoostedTrees =
       classificationEnsembleBoostedTrees;
72 trainedClassifier.ClassificationEnsembleRUSBoostedTrees =
       classificationEnsembleRUSBoostedTrees;
73 trainedClassifier.ClassificationEnsembleSubDiscriminant =
       classificationEnsembleSubDiscriminant;
74 trainedClassifier.ClassificationEnsembleSubKNN =
       classificationEnsembleSubKNN;
75 trainedClassifier.About = 'This struct is a trained classifier
       exported from Classification Learner R2016a.';
```

```
76 trainedClassifier.HowToPredict = sprintf('To make predictions on
       a new predictor column matrix, X, use: \n  yfit = c.
   predictFcn(X) \nreplacing ''c'' with the name of the variable
       that is this struct, e.g. ''trainedClassifier''. \n \nX must
       contain exactly 12 columns because this classifier was
   trained using 12 predictors. \nX must contain only predictor
   columns in exactly the same order and format as your training
       \ndata. Do not include the response column or any columns
   you did not import into \nClassification Learner. \n \nFor
   more information, see <a href="matlab:helpview(fullfile(
   docroot, ''stats'', ''stats.map''), ''
   appclassification_exportmodeltoworkspace'')">How to predict
   using an exported model</a>.');
77
78 % Extract predictors and response
79 % This code processes the data into the right shape for training
       the
80 % classifier.
81 % Convert input to table
82 inputTable = array2table(trainingData, 'VariableNames', {'
   column_1', 'column_2', 'column_3', 'column_4', 'column_5', '
   column_6', 'column_7', 'column_8', 'column_9', 'column_10', '
   column_11', 'column_12', 'column_13'});
83
84 predictorNames = {'column_1', 'column_2', 'column_3', 'column_4'
   , 'column_5', 'column_6', 'column_7', 'column_8', 'column_9',
       'column_10', 'column_11', 'column_12'};
85 predictors = inputTable(:, predictorNames);
86 response = inputTable.column_13;
87 isCategoricalPredictor = [false, false, false, false, false,
   false, false, false, false, false, false, false];
88
89 % Perform cross-validation
90
91 partitionedModel.baggedTrees = crossval(trainedClassifier.
   ClassificationEnsembleBaggedTrees, 'KFold', 20);
92 partitionedModel.boostedTrees = crossval(trainedClassifier.
   ClassificationEnsembleBoostedTrees, 'KFold', 20);
```

```
93 partitionedModel.rusBoostedTrees = crossval(trainedClassifier.
       ClassificationEnsembleRUSBoostedTrees, 'KFold', 20);
94 partitionedModel.subDiscriminant = crossval(trainedClassifier.
       ClassificationEnsembleSubDiscriminant, 'KFold', 20);
95 partitionedModel.subKNN = crossval(trainedClassifier.
       ClassificationEnsembleSubKNN, 'KFold', 20);
96
97
98 % Compute validation accuracy
99 validationAccuracy.baggedTrees = 1 - kfoldLoss(partitionedModel.
       baggedTrees, 'LossFun', 'ClassifError');
100 validationAccuracy.boostedTrees = 1 - kfoldLoss(partitionedModel
       .boostedTrees, 'LossFun', 'ClassifError');
101 validationAccuracy.rusBoostedTrees = 1 - kfoldLoss(
       partitionedModel.rusBoostedTrees, 'LossFun', 'ClassifError');
102 validationAccuracy.subDiscriminant = 1 - kfoldLoss(
       partitionedModel.subDiscriminant, 'LossFun', 'ClassifError');
103 validationAccuracy.subKNN = 1 - kfoldLoss(partitionedModel.
       subKNN, 'LossFun', 'ClassifError');
104
105
106 % Compute validation predictions and scores
107 [validationPredictions.baggedTrees, validationScores.baggedTrees
       ] = kfoldPredict(partitionedModel.baggedTrees);
108 [validationPredictions.boostedTrees, validationScores.
       boostedTrees] = kfoldPredict(partitionedModel.boostedTrees);
109 [validationPredictions.rusBoostedTrees, validationScores.
       rusBoostedTrees] = kfoldPredict(partitionedModel.
       rusBoostedTrees);
110 [validationPredictions.subDiscriminant, validationScores.
       subDiscriminant] = kfoldPredict(partitionedModel.
       subDiscriminant);
111 [validationPredictions.subKNN, validationScores.subKNN] =
       kfoldPredict(partitionedModel.subKNN);
112
113 confusionMat.baggedTrees=confusionmat(response,
       validationPredictions.baggedTrees,'order',[1 0]);
114 confusionMat.boostedTrees=confusionmat(response,
       validationPredictions.boostedTrees,'order',[1 0]);
```

```
115 confusionMat.rusBoostedTrees=confusionmat(response,
        validationPredictions.rusBoostedTrees,'order',[1  0]);
116 confusionMat.subDiscriminant=confusionmat(response,
        validationPredictions.subDiscriminant,'order',[1  0]);
117 confusionMat.subKNN=confusionmat(response,validationPredictions.
        subKNN,'order',[1  0]);
```