UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Renato Rodrigues da Silva

Desenvolvimento de toolbox de análise multivariada para o Matlab

Uberlândia, Brasil

2015

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Renato Rodrigues da Silva

Desenvolvimento de toolbox de análise multivariada para o Matlab

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: André Ricardo Backes

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2015

Renato Rodrigues da Silva

Desenvolvimento de toolbox de análise multivariada para o Matlab

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 05 de novembro de 2015:

André Ricardo Backes
Orientador

Mauricio Cunha Escarpinati

Pedro Moisés de Sousa

Uberlândia, Brasil
2015

Dedico este trabalho aos meus amigos e meus familiares pelo incentivo e apoio de sempre. Especialmente aos meus pais, que, mesmo estando longe, sempre estiveram ao meu lado, dando-me forças em todos os momentos.

Agradecimentos

Agradeço, em primeiro lugar, a Deus, que iluminou o meu caminho durante esta caminhada. Agradeço a todos os professores do curso, que foram muito importantes na minha vida acadêmica, principalmente, aos que foram verdadeiros mentores para mim: Mauricio Cunha Escarpinati, pelo seu companheirismo, amizade e todos os conselhos dados; e André Ricardo Backes, que, além de orientador, é um grande amigo e sempre foi peça fundamental nesta longa jornada de graduação e na realização deste trabalho de conclusão de curso. Agradeço, também, aos meus amigos e a todas as pessoas importantes pra mim e que sempre estiveram do meu lado, dando-me forças em todos os momentos, especialmente minha namorada e meus pais. Por fim, agradeço a todos aqueles que, de alguma forma, estiveram e estão próximos de mim, fazendo esta vida valer cada vez mais a pena.

"Embora ninguém possa voltar atrás e fazer um novo começo, qualquer um pode começar agora e fazer um novo fim." - Chico Xavier

Resumo

Apesar dos grandes benefícios do software Matlab, como a manipulação de matrizes, cálculos numéricos, interatividade e facilidade de criação de novos métodos, nota-se, nele, uma deficiência no que diz respeito a realização de testes estatísticos dos métodos desenvolvidos. Existem, hoje, algumas ferramentas livres que realizam testes como esses, entretanto, para a utilização dos dados gerados pelo Matlab nesses softwares, faz-se necessário que eles sejam convertidos para um formato em que possam ser reconhecidos e, então, salvos em disco. Esse processo pode acarretar erros de arredondamento e outros tipos de alterações nos dados que, ainda que sutis, podem comprometer a etapa de análise, levando a resultados diferentes dos esperados. O objetivo deste trabalho é desenvolver um pacote de métodos, ou uma *toolbox*, que tem como intuito suprir as carências do Matlab no que diz respeito à análise de dados em escala multivariada, trazendo um melhoramento ao software em relação à capacidade de realizar testes estatísticos, agrupamentos e classificação de dados.

Palavras-chave: Matlab, estatística, *toolbox*, classificação, agrupamento.

Lista de ilustrações

Lista de códigos fonte

Lista de abreviaturas e siglas

DTW	Dynamic Time Warping
EMD	Earth Mover's Distance
GPL	General Public License
KNN	K-Nearest Neighbors
NFL	Nearest Feature Line
RNA	Ribonucleic Acid
SFL	Shortest Feature Line

Sumário

1 Introdução

A utilização de métodos discriminantes é de grande importância em vários processos humanos, pois permite estabelecer uma diferenciação qualitativa e/ou quantitativa entre amostras de dados. Nos dias de hoje, em que o volume de informações é muito grande, essa classificação é uma boa prática para o melhoramento de processos que as utilizem como base.

Não só métodos discriminantes, mas também os estatísticos, em geral, implementados em programas de computador, servem de grande avanço para processos que necessitariam de muito trabalho manual ou que nem mesmo poderiam ser realizados dessa forma. Alguns exemplos desses métodos discriminantes são *Earth Mover's Distance* (EMD) e *Dynamic Time Warping* (DTW), que podem ser utilizados para comparações de padrões de fala e de cadeias de DNA/RNA.

O presente projeto teve como objetivo a implementação de uma *toolbox* (uma caixa de ferramentas) para o software Matlab, contendo métodos estatísticos que provessem, em uma escala multivariada, a capacidade de efetuar análise de dados em síntese estatística. Visto que ela é uma ciência dedicada à coleta, análise e interpretação de dados, utiliza as teorias probabilísticas para explicar a frequência de fenômenos e possibilitar a sua previsão no futuro.

O trabalho consistiu, inicialmente, no levantamento e estudo da base teórica estatística, isto é, o levantamento dos métodos e funções que seriam implementadas. Alguns exemplos são os discriminantes de Fisher, Bayes, KNN, além de métodos de agrupamento, como *k-means*. A segunda etapa consistiu em aplicar os métodos levantados à linguagem computacional, ou seja, desenvolver a teoria obtida durante o levantamento das técnicas estatísticas em linguagem nativa ao Matlab, compondo, assim, o pacote de programas.

Uma vez finalizada a implementação da *toolbox* de métodos estatísticos, eles podem ser utilizados nas mais diversas áreas do conhecimento que necessitem de classificação e comparações de dados em escalas multivariadas. Alguns exemplos são tarefas de comparações de cadeias de DNA/RNA e padrões de fala ou mesmo classificações de dados para separação de populações para armazenamento em bancos de dados ou previsões estatísticas para amostragem, proporcionando melhor eficácia e agilidade em processos realizados. Além disso, pode-se destacar a implementação de métodos recentes, como DTW, EMD, NFL e SFL, que possuem poucos precedentes em relação à nossa proposta de trabalho, assim como poucas referências para estudo. Métodos clássicos da estatística e que apresentam grande importância no contexto de classificação, como Fisher e Bayes, também foram implementados.

2 Referencial Teórico

Neste capítulo serão apresentados o embasamento teórico geral do trabalho e conceitos importantes para implementação do projeto, que serviram de base de estudos e de entendimento das teorias que seriam implementadas. Inicialmente, apresenta-se um histórico sobre análise estatística multivariada de dados, que, de uma maneira ampla, engloba e é base principal das teorias estatísticas. Em seguida, é apresentada uma introdução das bases estatísticas dos métodos importantes para este trabalho, que, além de servirem de base de estudo, foram também objetos de implementação deste projeto.

2.1 Análise multivariada de dados

A análise multivariada de dados é definida e caracterizada pela análise simultânea de múltiplas variáveis presentes em uma única relação ou em um conjunto de relações. Essa ciência auxilia na compreensão de comportamentos complexos de grupos de dados em ambiente afins e pode, dependendo da aplicação, acrescentar informações potencialmente úteis a eles, além de permitir preservar as correlações naturais entre as múltiplas influências de comportamento, sem isolar qualquer indivíduo ou variável (ANDERSON, 1958).

Em ambientes em que os "indivíduos" possuem características muito variadas, as múltiplas relações podem ser analisadas somente por meio da análise multivariada. Em vários ramos de trabalho, como o acadêmico, por exemplo, é necessário que o profissional seja capaz de sustentar sua análise de dados em bases teóricas e quantitativas. Esse é apenas um exemplo de uma das várias aplicações da análise multivariada de dados e de sua importância.

Principalmente nos dias de hoje, em que computadores possuem capacidade de processamento bem mais robusta que a dos existentes há relativamente pouco tempo atrás, pode-se notar um avanço no que diz respeito ao poder de processamento de dados e, consequentemente, análise multivariada de dados. Computadores podem fazer cálculos de grande quantidade de dados complexos e em tempo razoável, possibilitando processos que tomariam um tempo inimaginável ou que nem mesmo poderiam ser feitos se executados por força humana. O uso de computadores associados a programas estatísticos, tais como SPSS, SAS, R, Weka e Matlab, tem crescido consideravelmente e ganhado popularidade em diversas áreas pela facilidade de manipulação e aplicação de técnicas multivariadas.

Dentro da área de análise de dados, pode-se observar uma série de ramificações estatísticas. Uma das mais importantes áreas é denominada análise discriminante de dados,

que, inclusive, foi uma das mais utilizadas no processo de implementação da *toolbox* deste trabalho. Ela pode ser descrita como uma técnica multivariada de interdependência entre objetos, baseada na associação entre eles e um conjunto de características descritivas ou atributos especificados pelo manuseador dos dados, que utiliza a análise discriminante para prover classificação. Um dos objetivos da análise discriminante também é a redução dimensional da classificação dos objetos ou dados em conjunto de atributos representativos. Eles, então, passam a pertencer a determinada classe, com a qual os outros participantes partilham características em comum. Os passos para se chegar à classificação variam dependendo do método abordado e do fato de que podem levar em consideração duas ou mais classes (EATON, 1983).

Outra ramificação da análise estatística multivariada que merece destaque é a análise de agrupamento. É uma técnica cuja finalidade é agregar objetos com base nas características que eles possuem. O resultado são grupos que exibem máxima homogeneidade de objetos dentro deles e, ao mesmo tempo, máxima heterogeneidade entre eles. Em outras palavras, agrupamento de dados consiste em, basicamente, classificar uma amostra de dados ou objetos em grupos mutuamente excludentes com base na similaridade dos dados ou objetos, além de ser uma classificação de acordo com relações naturais dos dados, encontrando subgrupos significativos. Os grupos não são pré-definidos, são identificados durante o processo de análise, e, assim como na discriminante, os passos para o processo de agrupamento podem variar, porém todos os processos desse tipo levam algumas características básicas consideração. Elas são: medir a similaridade ou associação entre os dados para determinar o número de grupos, agrupá-los e, por fim, estabelecer um perfil dos dados (EVERITT; DUNN, 1991).

2.2 Métodos estatísticos de agrupamento e de classificação

Um exemplo de método discriminante é o KNN (*K-Nearest Neighbors*), ou os k vizinhos mais próximos. Na área de reconhecimento de padrões, esse algoritmo é um método não paramétrico usado para classificação e regressão. Em ambos os casos, a entrada consiste no conjunto de treino e de testes, além de a determinação de k mais próximos em um espaço característico(KELLER; GRAY; GIVENS, 1985). Na classificação, o dado ou objeto é classificado baseado em seus k vizinhos mais próximos, isto é, os dados que, a partir de uma determinada restrição, sejam os mais similares a ele. k é um número inteiro positivo, normalmente pequeno, então, se k é igual a 1, o objeto é simplesmente associado à classe daquele único vizinho mais próximo (MORENO-SECO; MICÓ; ONCINA, 2000). A saída do método depende se ele será empregado em regressão ou em classificação. Neste trabalho, em especial, ele foi empregado na forma de classificação ou discriminação e o retorno é a indicação da classe adequada aos objetos de entrada. Detalhes sobre sua aplicação e sua implementação poderão ser melhor observadas no próximo capítulo.

Outro exemplo importante de método discriminante é o de Fisher, também conhecido por método de Análise Discriminante Linear (do inglês: LDA). Ele é muito famoso, um dos mais utilizados no área da estatística e, assim como o KNN, permite a classificação de um conjunto de dados com base em um outro conjunto já classificado. Fisher possui duas variações, um dos métodos leva em consideração apenas duas classes e o outro pode distribuir os dados analisados em várias classes distintas. Além disso, LDA não exige hipóteses adicionais, ao contrário de outras técnicas que são baseadas em modelos probabilísticos (THEODORIDIS; KOUTROUMBAS, 2003).

Já o método ou teorema de Bayes, na área estatística e teoria probabilística, descreve a probabilidade do acontecimento de um determinado evento com base em condições, de certa maneira, relacionadas a ele. As probabilidades envolvidas nesse método podem ter diferentes aplicações, em certos casos, esse teorema é empregado diretamente como parte de inferência estatística. Por meio da interpretação de probabilidade Bayesiana, o teorema expressa a forma como um grau subjetivo de probabilidade deve, racionalmente, mudar ou não na relevância de suas evidencias (MALAKOFF, 1999). Um classificador Bayesiano considera que a informação importante está contida nas probabilidades condicionais das amostras, e não nos dados em si. Sua inferência possui aplicação em um grande leque de cálculos envolvendo probabilidades, não apenas os de inferência Bayesiana. Pode-se estabelecer que Bayes é um método classificador estatístico que atribui um certo dado a determinada classe, levando em consideração a probabilidade desse objeto pertencer a ela. Após o cálculo das probabilidades, obviamente, esse dado ou objeto é, então, alocado na classe em que a probabilidade dele pertencer seja maior(MOLINA, 1931).

O método de agrupamento *k-means* é um método de quantização vetorial, popular em mineração de dados e análise estatística. O algoritmo tem uma mera similaridade com o classificador KNN e é comumente confundido com ele devido ao k no nome. Ele visa particionar n observações de dados em k diferente grupos, de forma que cada observação pertença a um grupo em que os outros integrantes partilhem de características similares. Em outras palavras, *k-means* pode ser considerado um método de agrupamento não-hierárquico calculado por meio de repartição, formado por um procedimento que calcula os pontos que representam os "centros" dos grupos e que, então, espalha os dados homogeneamente e heuristicamente no conjunto de respostas obtidas até que se atinja um equilíbrio estático desses dados nos grupos (HAIR et al., 2005). Em seguida, o método faz uma divisão de todos os casos conseguidos pelos k grupos estabelecidos, e o critério escolhido para o cálculo é que definirá qual a melhor partição dos n casos. O procedimento começa utilizando valores dos primeiros casos dos k como uma estimativa temporária. Com a inclusão de cada dado no grupo, a média é alterada, mudando, assim, o valor do centroide. A cada passo, os dados são rearranjados ao grupo de centro mais próximo e as médias são novamente alteradas. Esse é um processo iterativo e que não termina até que as médias não sofram mais alterações ou que algum critério de número máximo de

iterações seja atingido (HAIR et al., 2005).

Outro método de agrupamento bastante usado é o chamado método de validação cruzada. Essa técnica é comumente utilizada em testes de eficiência de métodos classificadores, utilizando parte de sua própria amostra como treino, além de ser uma ferramenta poderosa de análise e um relevante recurso no desenvolvimento e ajuste de modelos de mineração de dados. Dentre as características do método, pode-se destacar: validação de robustez; avaliação de vários modelos de uma única instrução; construção de vários modelos; e identificação do melhor, baseado em estatísticas, por meio de comparações sucessivas e troca dos grupos em teste e treinamento. Assim, chega-se a um resultado final (SCHAFFER, 1993).

Além disso, *Nearest Prototype*, método de classificação, também denominado método de protótipos mais próximos, trata-se de um método simples e similar ao KNN, diferindo-se pelo fato de que ele utiliza um conjunto de amostras rotuladas, enquanto que o *Nearest Prototype* utiliza uma cadeia de protótipos (*prototypes*) para identificar a classe mais indicada a cada dado (BEZDEK; KUNCHEVA, 2001). Mais informações sobre o funcionamento desse método serão apresentadas na sessão de desenvolvimento.

Outro importante classificador é o NFL (*Nearest Feature Line*), que, traduzindo para o português, significa linha característica mais próxima. Assim como outros métodos discriminantes, ele tem como objetivo classificar uma determinada amostra de teste baseada em amostras de treino e, assim, encontrar a classe pertencente de cada dado. A razão de se utilizar NFL, uma vez que já existem outros métodos de classificação, é que alguns deles, como o KNN, não são paramétricos e, por isso, possuem taxa de erros, que é assintótica e maior que a do LDA, por exemplo, o que não acontece no NFL (DUDA; HART; STORK, 2001).

Já o método discriminante SFL (*Shortest Feature Line* Segment) ou menor segmento de linha característica, em português, é um classificador semelhante ao NFL, porém com a diferença de que, nele, a associação de um ponto ou um dado consiste em aplicar uma área de domínio a uma determinada linha característica. Essa área tem uma forma circular com seu comprimento e centro definido pelo ponto médio da FL (*Feature Line*) (HAN; HAN; YANG, 2011). Enquanto NFL tenta alocar dados levando em consideração apenas a linha característica mais próxima, esse processo não garante que os dados estejam realmente sendo alocados nas classes mais apropriadas. Uma linha formada por pontos distantes do dado que se deseja classificar pode ser traçada bem próxima a ele, fazendo com que o algoritmo cometa uma associação equivocada (HAN; HAN; YANG, 2011).

O Método discriminante DTW (*Dynamic Time Warping*) ou alinhamento temporal dinâmico trata-se de um método baseado em programação dinâmica comumente empregado em comparação de padrões de fala e sequências de DNA/RNA. Consiste em

realizar comparações entre duas séries temporais de dados de tamanhos não necessariamente iguais. Uma delas é utilizada com referência de comparação para a outra. Espera-se, como saída desse algoritmo, obter o coeficiente de alinhamento entre essas séries, isto é, uma representação do quão distantes estão as de teste e as de treinamento (FANG, 2009).

Por fim, o método EMD (*Earth Mover's Distace*), o mais recente estudado neste trabalho, tem como intuito medir a distância entre duas distribuições probabilísticas sobre uma determinada área. Ele assume uma dessas distribuições como referência e a outra como teste e calcula o quão distante uma está de se igualar à outra. Além disso, tem como ideia principal associar a distribuição de treino a uma massa de "terra". Partindo dessa premissa, o algoritmo visa expressar o custo de transporte dessa "terra", de modo a preencher as lacunas do treino até que elas se igualem à espalhada adequadamente no espaço e a distribuição de teste a um aglomerado de lacunas espalhadas no mesmo plano e espaço. O trabalho que se gasta para transportar "terra" da distribuição de treino a de teste, a fim de que a segunda se iguale à primeira, é denominado a distância que há entre elas, daí o nome EMD (RUBNER; TOMASI; GUIBAS, 2000). Esse método adota também a ideia de associar pesos aos pontos, que são próprios e não necessariamente iguais aos outros que compõem a distribuição.

3 Desenvolvimento

Nesta seção, será apresentada a rotina do desenvolvimento do projeto, bem como os detalhes da metodologia trabalhada. Uma vez que o presente estudo é um trabalho de implementação, a primeira parte do desenvolvimento consistiu em uma análise de programas estatísticos presentes no mercado e acessíveis ao meio acadêmico. Programas como Matlab, R e Weka foram analisados para compreender melhor o seu funcionamento e fazer um levantamento dos métodos estatísticos de análise multivariadas presentes neles e quais ainda necessitavam implementação, dando, assim, sentido à nossa proposta de contribuição. Em seguida, foi necessário um levantamento das técnicas estatísticas abordadas e um estudo mais aprofundado no entendimento desses métodos, buscando uma implementação perfeitamente condizente com as teorias. Esse processo de construção de conceitos também é mostrado neste capítulo, assim como a implementação desses métodos na linguagem de programação nativa ao Matlab, formando, assim, o resultado proposto do trabalho, a *toolbox* contendo as implementações desses métodos.

3.1 Reconhecimento do software Matlab e conceitos de R e Weka

O desenvolvimento do projeto iniciou-se com processo de reconhecimento dos softwares Matlab, R e Weka. O primeiro foi selecionado por ser o software escolhido para implementação do trabalho, enquanto R e Weka são outras ferramentas estatísticas com relevância ao tema.

Matlab tem o nome derivado da abreviação *Matrix Laboratory* e tem como especialidade a análise numérica, cálculo com matrizes e construção de gráficos em ambiente fácil de usar, em que problemas e soluções são expressos em uma forma quase que matematicamente legível. Além disso, várias funções estatísticas simples já se apresentam implementadas nele, tendo também uma vantagem sobre outras linguagens de programação no que diz respeito à velocidade de desenvolvimento.

Tomando como base as informações acima, vê-se o porquê da escolha desse software para as aplicações do projeto e tem-se noção de que seria imprescindível um domínio claro das suas funcionalidades para a elaboração de algoritmos que obedecessem sua sintaxe e, acima de tudo, aproveitassem ao máximo suas características específicas para apresentar alto desempenho.

R é o nome de um popular programa para cálculos estatísticos e geração de gráficos que está em uso por crescente número de analistas de dados, em empresas e no mundo acadêmico. Seu uso tem se tornado padrão, visto que os processos de mineração

de dados vivem uma era dourada, tanto para determinar preços de publicidade e descobrir novos medicamentos de forma mais rápida, quanto para fazer a sintonia fina de modelos financeiros, por exemplo (TORGO, 2009). Empresas das mais diversas, tais como *Google*, *Pfizer*, *Merck*, *Bank of America* e *Shell*, usam o R. Esse software também encontrou rápida aceitação entre os estatísticos, engenheiros e cientistas, que não conhecem bem a programação de computadores e o consideram fácil de usar, já que ele permite que os estatísticos realizem análises complexas e detalhadas sem que precisem conhecer em detalhe o funcionamento dos sistemas de computação (CHAMBERS, 2008).

Já o software Weka (*Waikato Environment for Knowledge Analysis*) começou a ser escrito em 1993, usando a linguagem Java, na Universidade de Wakato, Nova Zelândia, sendo adquirido posteriormente por uma empresa no final de 2006. Ele encontra-se licenciado sob abrigo da *General Public License* (GPL), sendo, portanto, possível estudar e alterar o respectivo código fonte. Tem como objetivo agregar algoritmos provenientes de diferentes abordagens/paradigmas na subárea da inteligência artificial, dedicada ao estudo da aprendizagem por parte de máquinas. Essa subárea pretende desenvolver algoritmos e técnicas que permitam a um computador "aprender", no sentido de obter novo conhecimento, quer indutiva ou dedutivamente. O Weka procede a análise computacional e estatística dos dados fornecidos a partir das técnicas de *data-minning*, tentando, indutivamente, a partir dos padrões encontrados, gerar hipóteses para soluções e inclusive teorias sobre os dados em questão (WITTEN; FRANK, 2005).

O presente estudo começou com leitura de livros e apostilas com foco mais básico e destinado a iniciantes na programação Matlab, com posterior aplicação desse conhecimento na elaboração de programas simples. Com o passar das semanas, o acúmulo de conhecimento possibilitou a elaboração de programas mais aprimorados e com melhor aproveitamento de recursos próprios do software. Por exemplo, programas antes feitos com utilização de laços foram substituídos por vetorização, que, além de ser mais legível, é bem mais eficiente. A utilização de códigos procedimentais foi otimizada com a utilização de funções já existentes no Matlab, por exemplo, a função *unique*, que serve para eliminar elementos repetidos em um vetor, e que, após ter sido descoberta, substituiu a que havia sido implementada, apresentando, inclusive, a funcionalidade extra de retorno de vetor de índices. A criação de algumas funções foi dispensada com a descoberta de algumas similares já implementadas no Matlab. Um exemplo é a de cálculo da distribuição normal, que se encontra na base de funções com o nome de *normpdf* e é expressa pela seguinte fórmula:

$$p(x|\mu_i, S_i) = \frac{1}{\sqrt{(2\pi)^p |S_i|}} exp[\frac{-1}{2}(x-\mu)^T S_i^{-1}(x-\mu_i)]$$

em que μ_i é a média e S_i é a matriz de covariância de certa classe.

Outra parte importante do reconhecimento foi o levantamento de funções existentes em softwares similares ao Matlab, tanto as que o projeto visa implementar, quanto as outras formas de análise, sejam essas funções de discriminação ou de agrupamento.

3.2 Levantamento das bases estatísticas e das técnicas a serem implementadas

O levantamento das bases estatísticas consistiu no estudo aprofundado dos teoremas que compõem os principais métodos, como os de agrupamento e discriminação. O aprofundamento desses métodos ocorreu de acordo com a forma com que eles foram sendo implementados no software. Assim, uma nova função só foi iniciada quando a anterior se encontrava totalmente pronta, se houvesse dependência entre elas.

Dado que o objetivo da estatística é a produção da melhor informação possível a partir dos dados disponíveis, teve-se a preocupação de ter bem definido em mente o que se queria fazer e com a certeza de estar bem entendido, para que os programas refletissem o que a teoria em si realmente falava .

3.2.1 Medidas estatísticas básicas

Algumas medidas estatísticas básicas já se encontram implementadas no Matlab e são muito utilizadas por várias funções mais complexas, inclusive as que o projeto vem abrangendo. Pode-se destacar as mais utilizadas:

- **Média aritmética**

  É representada pelo símbolo μ e tem como conceito esboçar um valor central ou um valor médio de determinados valores tomados. Por exemplo, tendo-se n valores, a média aritmética deles é obtida por meio do somatório de todos os elementos, seguido da divisão pelo valor n, que representa o número de elementos:

$$\mu = \frac{x_1 + x_2 + x_3 + \cdots + x_n}{n}$$

- **Desvio Padrão**

  É representado pelo símbolo σ (sigma minúsculo) e é uma medida de dispersão dos valores de uma distribuição normal em relação a sua média. É definido pela fórmula:

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{n}}$$

- **Variância**

  A variância tem como objetivo analisar o grau de variabilidade ou a dispersão de dados. Em outras palavras, por meio dela, pode-se perceber o quão longe um valor se encontra do esperado. A fórmula da variância é dada por:

$$\sigma^2 = \frac{\sum(x_i - \mu)^2}{n}$$

3.2.2 Z-Score

Também conhecido como *score* padronizado, ajuda a entender onde determinado elemento se encontra em relação aos demais em uma distribuição e indica o quanto, acima ou abaixo da média, ele está em termos de unidades de desvio. O z-score é calculado usando a média e o desvio padrão dos dados.

A utilização do *z-score* consiste na padronização de *scores*, que é o processo de converter o *score* bruto de uma distribuição em *score* z. O bruto é o valor individual observado emuma determinada variável de medição. Sua fórmula é dada como sendo:

$$z = \frac{x - \mu}{\sigma}$$

em que x representa o score bruto.

3.2.3 Método classificador KNN- *K Nearest Neighbours*

Esse método de classificação é bastante usado em processos de reconhecimento de padrões. Apesar de ter sido fundamentado por volta dos anos 50, só ganhou popularidade com o surgimento de computadores mais potentes. O processo de classificação se dá pelos seguintes passos:

1. Tome como base um conjunto X de elementos. Cada elemento x_i de X é um conjunto de dados x_i= $(x_{i1}; x_{i2}, \ldots, x_{in})$ já classificado em uma determinada classe $c_i = 1, 2, \ldots, M$, em que M é o número de classes;

2. Tome também outro conjunto de dados Y, em que cada elemento é representado por $y_j = (y_{j1}, y_{j2}, \ldots, y_{jn})$ o qual se deseja classificar;

3. O método KNN busca classificar o conjunto Y, calculando as distâncias de cada elemento de Y para todos os do grupo de treinamento X. Para cada elemento de Y, é calculada a distância a cada um de X. Para melhor representação, considere que, para cada y_i, existirá um vetor com as distância deles a todos os elementos de X. Assim, para y_1, por exemplo, existiria $D_1 = (d_1, d_2, \ldots, d_n)$, e para y_n, $D_n = (d_1, d_2, \ldots, d_n)$;

4. Tomando-se o vetor das distâncias D_i de um elemento y_i, considera-se os k elementos de treinamento mais próximos de y_i, ou seja, os que contenham menores distâncias ou os menores valores do vetor de distâncias. Dentre esses k conjuntos, verifica-se qual a classe que aparece com mais frequência. O elemento y_i será classificado dentro dessa classe;

5. Em caso de empate de maior frequência na classificação de algum elemento de Y, o quarto passo é refeito, considerando-se os $k-1$ vizinhos mais próximos. Em caso de novo empate, o processo é refeito até que a escolha seja unânime.

3.2.4 Método discriminante de Fisher

Esse é um dos métodos de classificação mais famosos e um dos mais utilizados no meio estatístico. Assim como o KNN, ele permite classificar um certo conjunto de dados, baseado em um outro já classificado (THEODORIDIS; KOUTROUMBAS, 2003).

3.2.4.1 Método discriminante de Fisher para duas classes

Inicialmente, o conceito estudado trata apenas de classificação baseado em duas classes, especificamente. O processo de classificação se dá por meio dos seguintes passos:

1. Tome um conjunto X com elementos classificados em somente duas classes diferentes e Y um conjunto de dados a se classificar;

2. Inicialmente, é calculado a matriz de médias aritméticas e a matriz de covariâncias de cada umas das duas populações de X (μ_1, μ_2 e S_1, S_2);

3. Em seguida, é calculada a matriz S_p, que é a combinação linear das duas matrizes de covariância, representada pela fórmula:

$$S_p = \frac{1}{2}S_1 + \frac{1}{2}S_2$$

4. Em seguida, é necessário achar o coeficiente m, que representa o ponto médio entre as duas médias univariadas amostrais, representado por:

$$m = \frac{1}{2}(\mu_1 - \mu_2)^T S_p^{-1}(\mu_1 - \mu_2)$$

5. Na prática, a função discriminante de fisher é dada pela fórmula:

$$f = (\mu_1 - \mu_2)^T S_p^{-1} r$$

6. Como já citado, μ_1 e μ_2 equivalem às médias, enquanto S_p equivale à matriz da combinação linear das matrizes de covariância das duas populações. Portanto, para se completar os dados da equação, é necessário inserir à variável r um valor y_i do conjunto Y. O coeficiente m é utilizado como elemento comparativo, se f for menor que m, y_i é alocado na classe que representa a população 2, caso contrário, é alocado na classe que representa a população 1. O processo é repetido para todos os elementos do conjunto Y.

3.2.4.2 Método discriminante de Fisher para várias classes

Como descrito na seção anterior, inicialmente, esse método tratava-se apenas de classificação baseada em duas classes. Nessa nova etapa, a implementação consistiu um método similar, porém que permitisse a classificação de uma amostra em mais do que duas classes. Além disso, não é necessário que os dados classificados sigam uma distribuição normal, entretanto, é imprescindível que as matrizes de covariâncias populacionais sejam iguais dentro dessa amostra de treinamento (THEODORIDIS; KOUTROUMBAS, 2003).

O método de classificação para várias classes consiste basicamente nos seguintes passos:

Tomar um conjunto X com elementos classificados em classes g diferentes e Y um conjunto de dados a se classificar.

1. Inicialmente, é calculado o vetor de médias aritméticas de cada uma das g classes, $\bar{\mu}$, assim como suas g matrizes de covariâncias $\sum$;

2. Calcular a matriz B, que basicamente representa a soma de produtos cruzados das g classes, ou seja,

$$B_{gxg} = \sum_{i=1}^{g} \sum_{j=1}^{n_i} n_i(\bar{\mu}_i - \bar{\mu})(\bar{\mu}_i - \bar{\mu})^T$$

   Note que se os vetores médias forem iguais, não há diferença entre as classes, logo $B = 0$;

3. Calcular a matriz W de soma de quadrados:

$$W_{gxg} = \sum_{i=1}^{g} \sum_{j=1}^{n_i} n_i(\bar{x}_{ij} - \bar{x}_l)(\bar{x}_{ij} - \bar{x}_l)^T$$

   Em que n_i representa o número de elementos em cada classe do treino;

4. Calcular S_p, que representa o estimador de $\sum$.

$$S_p = \frac{1}{n_1 + n_2 + \cdots + n_g - g} W$$

5. Calcular os autovetores da matriz Q, definida por:

$$Q = S_p^{-1} B_0$$

6. Tomar um elemento y_i que se deseja classificar do conjunto de testes Y e, para cada classe, calcular:

$$f = \sum_{j=1}^{s} l_j^T (y_i - \mu_k)^2$$

Em que l_j^T é cada autovetor de Q no intervalo s. k representa cada uma das g classes, enquanto s representa $min(g-1, dim)$, ou seja, o valor mínimo entre o número de classes menos 1 e a dimensão das variáveis de treino;

7. O resultado do cálculo anterior para cada classe k é um vetor de tamanho s com os valores de f. Deve-se calcular, então, o somatório desse vetor;

8. Tomando o novo vetor com os g somatórios. Verificar qual o menor e alocar y_i na classe a qual se refere esse valor;

9. A matriz de teste estará classificada após ser realizado esse processo para todos os elementos y_i.

3.2.5 Método discriminante de Bayes

O método discriminante de Bayes é um classificador estatístico que atribui um objeto em uma determinada classe, baseando-se na probabilidade dele pertencer a ela. Após o cálculo das probabilidades, o objeto é alocado na classe cuja probabilidade dele pertencer é maior.

Um classificador Bayesiano considera que a informação importante está contida nas probabilidades condicionais das amostras, e não nos dados em si (MOLINA, 1931). É considerado um classificador ótimo se conhecermos a função de densidade probabilidade da teoria de Bayes que gera os dados, definida por:

$$P(C|Y) = \frac{P(Y|C)P(C)}{P(Y)}$$

Na prática, isso não ocorre tão facilmente e é necessário buscar soluções alternativas. Uma possibilidade é o uso do cálculo de distribuição normal, que é dado pela equação:

$$P(Y|C) = p(y|\mu_i, S_i) = \frac{1}{\sqrt{(2\pi)^p |S_i|}} exp[\frac{-1}{2}(y-\mu)^T S_i^{-1}(y-\mu_i)]$$

em que μ_i é a média e S_i a matriz de covariância da classe em questão.

Essa é, inclusive, a solução utilizada no projeto, com o emprego da função *normpdf*, já citada. Apesar de ser uma boa alternativa para o cálculo da densidade probabilidade bayesiana, o projeto visa fazer o levantamento e o estudo das demais possibilidades.

Por fim, a população é classificada na classe que apresenta maior produto entre a probabilidade a priori e o resultado da equação de distribuição normal.

Exemplificando,

1. Tome como base um conjunto X de elementos. Cada elemento x_i de X é um conjunto de dados $x_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ já classificado em uma determinada classe $c_i = 1, 2, \ldots, M$, em que M é o número de classes;

2. Para cada uma, é necessário que se calcule a média aritmética e a matriz de covariância de seus elementos, e que se obtenha também a probabilidade a priori de cada classe ocorrer;

3. Atribuindo os valores à equação da distribuição normal, cada elemento de Y é substituído na variável da equação, enquanto os outros dados (média aritmética e matriz de covariância) pertencem à população da classe em que se deseja calcular a probabilidade. Lembrando que existem M classes, essa operação vai ser realizada M vezes para cada elemento de Y, produzindo um vetor de resultados tamanho M, em que cada um deles é procedente da equação;

4. Cada elemento desse vetor deve ser multiplicado com a probabilidade a priori de cada classe ocorrer, o resultado é um novo vetor de probabilidades. Em outras palavras, cada elemento desse novo vetor pode ser representado pela fórmula:

$$P(C|Y) = P(Y|C)P(C)$$

   Simplificando, a probabilidade de uma classe c_i ser atribuída a um elemento y_j equivale ao produto da densidade probabilidade (que foi substituída pela de distribuição normal) e a probabilidade a priori;

5. Já que os elementos desse novo vetor representam as probabilidades de ocorrência da classe, o maior indica a maior probabilidade de uma população pertencer à determinada classe. Portanto, a classe que representa esse maior valor é a atribuída ao elemento y_i. Esse processo é realizado para cada elemento de Y.

Nota: a probabilidade a priori é a probabilidade de determinada classe ocorrer antes mesmo de se observar os dados que se deseja classificar. Ela é encontrada por meio da operação:

$$P(c_i) = \frac{N_i}{N}$$

em que N_i é a quantidade de elementos pertencentes à classe c_i e N é a quantidade total de elementos do conjunto.

3.2.6 Método de agrupamento *k-means*

O método de *k-means*, ou, em português, k-médias, é um método de clusterização ou de agrupamento que está entre os mais conhecidos e mais utilizados em problemas práticos da estatística. Basicamente, cada elemento amostral é alocado ao grupo cujo centroide (vetor de médias amostral) é o mais próximo do vetor de valores observados para o respectivo elemento. Originalmente, o método é composto por quatro passos:

1. Primeiramente, escolhe-se k centroides, chamados de "sementes" ou "protótipos" para se inicializar o processo de partição;
2. Cada elemento do conjunto de dados é, então, comparado com cada centroide inicial, por meio de uma medida de distância que, em geral, é a distância euclidiana. O elemento é alocado ao grupo cuja distância é menor;
3. Em seguida, recalcula-se os valores dos centroides e repete-se o passo 2 para os novos centroides, considerando os desses novos grupos;
4. Esse processo se repete até que todos os elementos amostrais estejam "bem alocados" em seus grupos, isto é, até que nenhuma realocação de elementos seja necessária e a atual fique igual à anterior.

O Matlab já possui esse método em sua base de funções, entretanto, outros softwares estatísticos podem utilizar formas diferentes de implementação do *k-means*. A escolha das sementes iniciais influencia no resultado final de agrupamento, portanto, é necessário cuidados no momento da escolha. Das formas de seleção, utiliza-se mais as de: escolha aleatória, escolha prefixa, escolha dos k primeiros valores do banco de dados, uso de técnicas hierárquicas aglomerativas e escolha via uma variável aleatória.

3.2.7 Método de validação cruzada

Essa técnica é bastante utilizada para testar a eficiência de métodos de classificação. Ela visa mostrar o quanto uma classificação foi eficaz, utilizando parte de sua própria amostra como treinamento.

O método, primeiramente, divide seu conjunto ou campo de amostra em certo número de partições (um subconjunto de um determinado conjunto de dados). Em seguida,

utiliza uma dessas partições como dados de treino e o restante como dados de validação ou teste. Por meio de comparações sucessivas e troca das partições em teste e treino, é possível chegar a um resultado. Por meio dele é que se sabe o quão eficaz o método testado é (SCHAFFER, 1993).

Após o particionamento dos conjuntos, é realizado o processo de comparações sucessivas, que resultará em uma estimativa do quão eficaz é um método de classificação testado. Quanto maior esta estimativa de acertos, menor será a quantidade de erros de predição, ou classificação.

O Matlab apresenta, implementado em suas versões mais recentes, as funções *crossvalind* e *cvpartition*, que realizam esse processo. Um dos parâmetros de entrada deles é o método de particionamento para a validação cruzada, que pode ser qualquer um dos três descritos acima.

A principal diferença entre essas duas funções é que a *cvpartition* tem, como saída, um objeto construído a partir das informações dadas como entrada (método de agrupamento, números de partições e conjunto de dados), por padrão *cvpartition*, considera $k = 10$, em caso da informação sobre o número de partições estar ausente. Esse objeto de saída visa à validação cruzada e tem como atributos o tamanho do conjunto de amostra, a quantidade k de partições divididas e o tamanho do subconjunto de treinamento e o do subconjunto de teste.

Já a função *crossvalind*, parece ser mais flexível, retornando mais de um parâmetro. Ela gera aleatoriamente um vetor de índices para o método de validação escolhido e atua sobre determinado número de observações. k (quantidade de partições para divisão), por padrão, é 5, quando ausente. A saída também pode, além desse vetor, produzir outro referente ao teste, já que o primeiro referia-se ao treinamento.

Existem diversas formas de realizar o particionamento dos dados, sendo as três mais utilizadas: o método *holdout*, *k-fold* e *leave-one-out*.

- **Método holdout**

  Esse método consiste em dividir um conjunto de dados em duas partições, ou subconjuntos, um para treinamento e outro para teste. A quantidade de dados em cada partição não precisa ser necessariamente igual. Uma proporção aconselhável é a utilização de 2/3 dos dados como treinamento e 1/3 para teste.

  Esse tipo de particionamento é indicado para conjuntos de dados mais extensos, pois, quando utilizado em pequenos conjuntos, o erro calculado na previsão de classes pode sofrer muita variação.

- **Método *k-fold***

O método *k-fold* consiste em particionar o conjunto de dados em k partições, em que cada uma delas apresente o mesmo número de dados. São realizadas k iterações e, pra cada uma delas, um dos k subconjuntos é utilizado como teste, enquanto os $k-1$ restantes representam os dados de treinamento.

- **Método *leave-one-out***

  Ele particiona o conjunto de dados em uma quantidade n, em que essa quantidade é igual ao número de dados do conjunto. É criada uma partição para cada dado. Esse método pode ser considerado um tipo específico de *k-fold* em que k equivale à quantidade de elementos do conjunto de dados.

  O método apresenta uma investigação mais apurada no momento dos testes, porém seu custo computacional é muito grande, se comparado aos outros e é extremamente recomendado para conjuntos que apresentam poucos dados.

3.2.8 Método classificador *Nearest Prototype*

A tradução mais adequada para esse método classificador seria método de protótipos mais próximos. Ele é bastante simples e muito parecido com o KNN (k vizinhos mais próximos). A principal diferença entre eles é o fato de que, ao invés de usar um conjunto de amostras rotuladas, o *Nearest Prototype* utiliza uma cadeia de protótipos, ou *prototypes*, para identificar a qual classe pertence cada elemento da amostra (BEZDEK; KUNCHEVA, 2001).

O método de classificação pode ser utilizado para várias classes e consiste basicamente nos seguintes passos:

Tome um conjunto X com elementos classificados em g classes diferentes e Y um conjunto de dados a se classificar.

1. Inicialmente, calcule o vetor de médias aritméticas de cada uma das g classes, $\bar{\mu}$;

2. Do conjunto Y, tome um elemento y_i e uma das g classes calcule e o protótipo μ_l. No teste haverá l protótipos:

$$\mu_l = \frac{1/||y_i - \mu_k||^{2/(m-1)}}{\sum_{j=1}^{g}(1/||y_i - \mu_j||^{2/(m-1)})}$$

   Em que μ_j e μ_k representam os valores de média de cada uma das classes g. $||.||$ representa a distância (Euclidiana, por exemplo) entre eles e cada elemento ou amostra do conjunto de teste é representado por y_i. m geralmente é 2;

3. Alocar y_i na classe associada ao maior valor dentre os protótipos obtidos;

4. O conjunto de dados Y estará classificado após ser realizado esse processo para todos os elementos y_i.

3.2.9 Método classificador NFL- *Nearest Feature Line*

Em português, é chamado de linha característica mais próxima. Com o desenvolvimento da classificação de padrões, vários métodos de classificação têm sido propostos. Assim como KNN, esse método visa classificar uma amostra de teste baseada em uma já feita (treino) em várias classes g. Apesar de algumas semelhanças, a principal diferença entre eles é que KNN, apesar de eficaz, é um método não paramétrico que possui taxa de erro assintótico, que é, no máximo, duas vezes a taxa de erro de Bayes (DUDA; HART; STORK, 2001).

A rotina de classificação por linha mais próxima consiste basicamente em traçar retas ligando as amostras de treinamento de uma determinada classe, duas a duas, sem repetição de pares. A Figura 1 representa um exemplo desse cenário. Cada uma das amostras de teste terá sua distância medida a cada uma das linhas de características. A que apresentar a menor distância à amostra tomada atribuirá sua classe a ela (HAN; HAN; YANG, 2011).

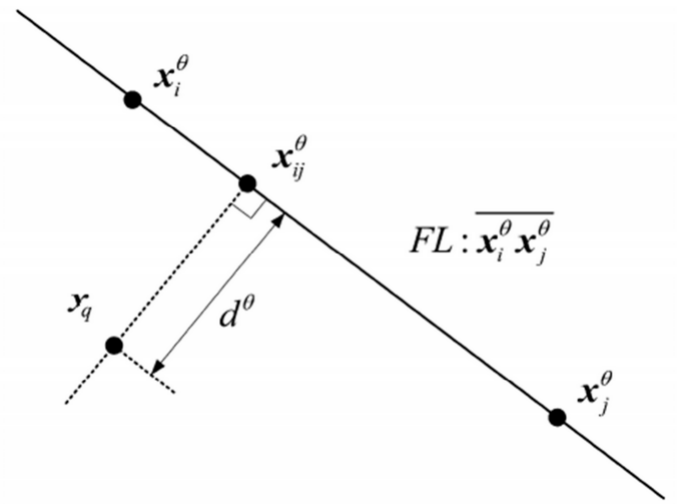

Figura 1 – Representação de ligação de amostras por linha FL. Extraída de (HAN; HAN; YANG, 2011)

O método pode ser melhor compreendido seguindo os seguintes passos:

1. Tomado um elemento y_i que se deseja classificar do conjunto Y, para cada uma das g classes, haverá uma quantidade n de amostras. Assumindo que as amostras dentro de uma mesma classe são agrupadas duas a duas sem repetição, pode se dizer que a quantidade de retas formadas é igual à combinação de n, tomando duas a duas

$(C_n, 2)$, o que resultará em uma quantidade m de retas. A equação matemática que representa cada reta consiste em:

$$x_{ij}^{\theta} = (1 - \mu)x_i^{\theta} + \mu x_j^{\theta}$$

Em que o coeficiente μ é representado por:

$$\mu = \frac{((y_i - x_i^{\theta})^T(x_j^{\theta} - x_i^{\theta}))}{((x_j^{\theta} - x_i^{\theta})^T(x_j^{\theta} - x_i^{\theta}))}$$

2. Calcular a distância entre a amostra y_i e cada uma das retas x_{ij}^{θ}. A métrica usada é a Euclidiana representada por:

$$d(y_i, \overline{x_l^{\theta} x_j^{\theta}}) = ||y_i - x_{ij}^{\theta}||$$

3. Ao final dos cálculos, nota-se que a distância foi calculada q vezes, que consiste na soma do coeficiente m de cada uma das classes;

4. y_i será alocado em uma das q classes que corresponda aos pontos que formam a reta mais próxima a ele, ou seja, que apresente menor distância euclidiana;

5. A matriz de teste estará classificada após ser realizado esse processo para todos os elementos y_i.

Com o conceito do NFL bem definido, pode-se destacar duas desvantagens do KNN em relação a ele: a capacidade de representação do conjunto e a taxa de erro dependem de como as amostras de treinamento e teste são escolhidas quanto às possíveis variações. O desempenho do KNN também depende da definição da métrica de distância usada, por exemplo a Euclidiana. Em contrapartida, NFL possui maior custo computacional, por ter maior complexidade em seu algoritmo, embora tenha a vantagem de ser mais apurado.

3.2.10 Método discriminante SFL- *Shortest Feature Line Segment*

Em português, menor segmento de linha característica, esse método de classificação é muito semelhante ao NFL, porém foi desenvolvido com a missão de corrigir o seguinte erro do NFL: o fato de ele alocar um ponto à classe referente à linha característica mais próxima não quer dizer que o ponto realmente esteja sendo alocado na classe mais apropriada. Uma linha formada por pontos longínquos a um que se deseja classificar pode ser traçada bem próxima a ele, mesmo os extremos estando distantes. Pontos mais próximos a ele podem formar uma linha um pouco mais distante, fazendo com que o algoritmo cometa um equivoco. Observando a Figura 2, é possível visualizar esse cenário. Pode-se notar que a linha característica (FL) mais próxima à x_q é formada por dois pontos

extremos x_1^c e x_2^c, que não são os mais próximos ao ponto. A proposta desse novo método é resolver esse problema, fazendo com que x_q seja associado à classe que realmente está mais próxima.

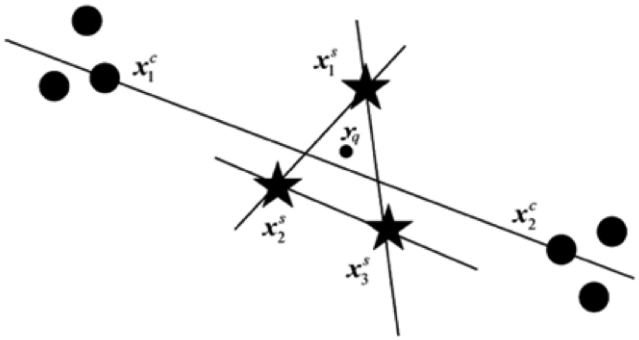

Figura 2 – representação de uma classificação inadequada, usando como base somente a distância entre o ponto e a linha de característica NFL. Extraída de (HAN; HAN; YANG, 2011)

A associação justa de um ponto, proposta pelo método SFL, consiste em aplicar uma área de domínio a uma determinada linha característica. Essa área tem uma forma circular com seu comprimento e seu centro definido pelo ponto médio da FL, sendo que um ponto só pode ser associado à classe de uma linha se este estiver dentro da área de domínio da FL e se não houver uma classe mais próxima a ela. Além disso, ele se encontra dentro de uma área de domínio se o seu ângulo, associado aos pontos extremos da reta, formar no mínimo 90°. A Figura 3 mostra os três casos em que um ponto pode estar posicionado a uma determinada FL. Como o mostrado na Figura 3(c), ele não pode ser associado à classe dos pontos extremos da FL, diferentemente dos casos (a) e (b), em que está dentro da área de representação ou exatamente na linha.

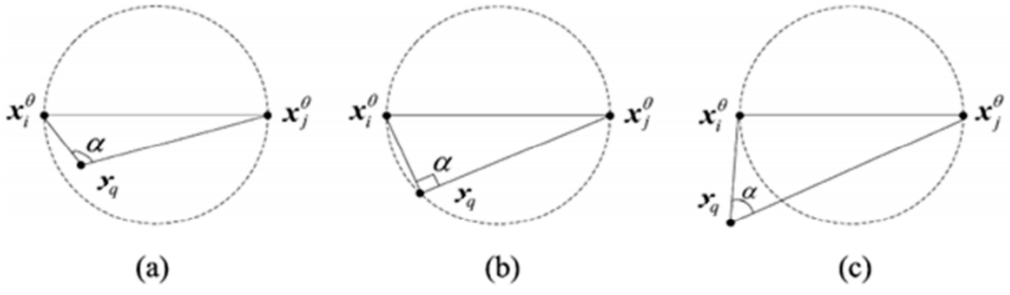

Figura 3 – representação de possíveis posicionamentos de um ponto. presente em (HAN; HAN; YANG, 2011)

O processo de classificação pode ser definido pelos seguintes passos:

1. Do conjunto de teste Y, tomar um elemento y_i que se deseja classificar. Para cada uma das g classes, haverá uma quantidade n de amostras. Assumindo que as amos-

tras dentro de uma mesma classe são agrupadas duas a duas sem repetição, pode-se dizer que a quantidade de retas formadas é igual à combinação de n tomado duas a duas ($C_{n,2}$), que resultará em uma quantidade m de retas.

3.2.11 Método discriminante DTW- *Dynamic Time Warping*

A tradução mais adequada desse método para português seria "alinhamento temporal dinâmico". Trata-se de um algoritmo baseado em programação dinâmica muito utilizado atualmente para comparação de padrões de fala e de sequências de DNA/RNA. Consiste basicamente em fazer comparações entre duas séries temporais de dados que não precisam necessariamente ser do mesmo tamanho. Uma delas é utilizada como modelo de referência com o qual a série a ser verificada será comparada. Ao final da computação do algoritmo, espera-se obter o coeficiente de alinhamento entre essas séries, ou seja, uma representação do quão distante a de teste está de se alinhar com a de treinamento (FANG, 2009). A Figura 4 mostra o alinhamento resultante entre duas séries temporais hipotéticas desalinhadas, R e V.

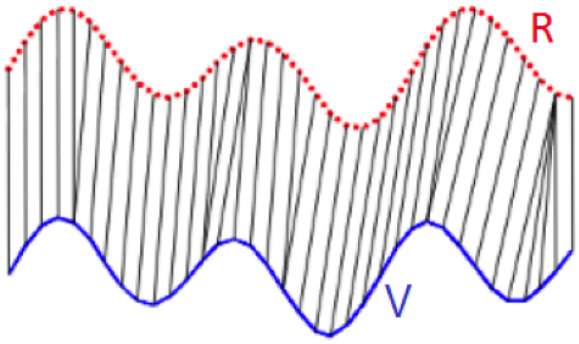

Figura 4 – Alinhamento resultante entre duas séries temporais hipotéticas, R e V.

Esse método possui a vantagem de ser bastante simples, não exigindo uso de equações matemáticas complexas, apenas o conceito de cálculo de distância associado à minimização de resultado, utilizando a programação dinâmica. O processo de comparação é descrito nos seguintes passos:

1. Considere dois vetores ou sequências temporais de dados, R e V, que representam respectivamente dados de referência ou treino e dados de teste. Esses dois conjuntos não precisam necessariamente ser do mesmo tamanho;

2. Monta-se uma matriz contendo as distâncias acumuladas entre os pontos, DA (Figura 5). Essa matriz possui tamanho n-por-m, em que n e m são os tamanhos dos vetores R e V, respectivamente. Usualmente, coloca-se o menor padrão no eixo Y da

matriz, deixando o maior ao longo do eixo X. O primeiro elemento é preenchido com a distância euclidiana absoluta entre os primeiros elementos de cada vetor, $|r_1 - v_1|$;

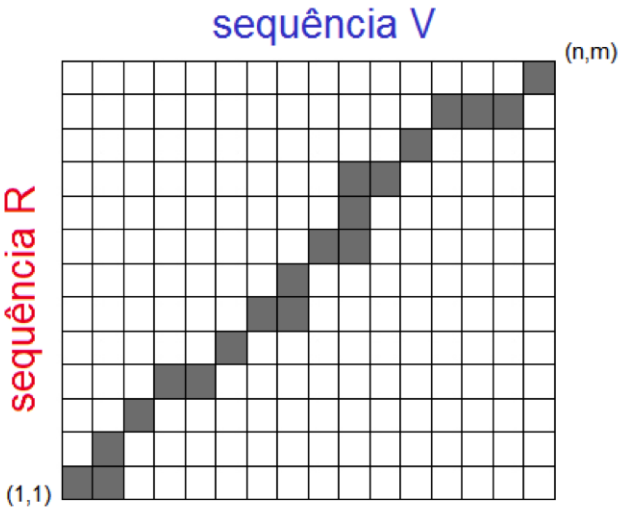

Figura 5 – exemplo de matriz contendo as distâncias acumulada entre as duas sequências hipotéticas R e V, e um caminho W.

3. O restante da primeira linha é preenchido pela soma entre a distância absoluta dos elementos daquela posição de cada vetor com o valor do elemento da coluna anterior à esquerda. A equação que representa o valor de cada célula dessa linha é definida por:

$$DA[0, j] = |R_i - V_j| + DA[0, j - 1]$$

4. Após o preenchimento da primeira linha. Deve-se fazer o preenchimento dos demais elementos da primeira coluna. A lógica é bem parecida, basta calcular, para cada item, a soma da distância absoluta dos elementos daquela posição, de cada vetor, com o valor do elemento da linha anterior, como abaixo:

$$DA[i, 0] = |R_i - V_j| + DA[i - 1, 0]$$

5. Nesse ponto, a matriz conterá a primeira linha e coluna preenchidas. Já que a primeira célula preenchida foi a de índices (1, 1), que se encontra na extremidade inferior esquerda, as demais células da matriz são preenchidas da esquerda pra direita e de baixo para cima, somando o valor da distância absoluta dos elementos dos vetores na posição da matriz que se deseja preencher a célula com o menor valor dos primeiros elementos acima, à direita e à diagonal:

$$DA[i, j] = |R_i - V_j| + min(DA[i, j - 1], DA[i - 1, j], DA[i - 1, j - 1])$$

6. Depois de preenchida toda a matriz, o próximo passo é encontrar o conjunto contíguo de elementos da que defina o mapeamento entre R e V. Esse conjunto é um "caminho" percorrido na matriz que podemos chamar de W. Um elemento k de W é definido como $w_k = (i, j)_k$. Esse caminho, porém, está sujeito a algumas restrições:

   - **Limites:** Requer que o caminho W comece e termine em cantos diagonalmente opostos ao da matriz. Então, $w_1 = (1, 1)$ e $w_k = (n, m)$;
   - **Continuidade:** Tomando $w_k = (a, b)$, então $w_{(k-1)} = (c, d)$, em que $a - b \leq 1$ e $b - d \leq 1$. Essa propriedade garante que o caminho seguido sempre terá células adjacentes (inclusive as diagonalmente adjacentes);
   - **Monotonicidade:** Tomando $w_k = (a, b)$, então $w_{(k-1)} = (c, d)$, em que $a - c \geq 0$ e $b - d \geq 0$. Essa propriedade garante que o caminho seguido sempre terá células monotonicamente espaçadas no tempo.

7. Existem vários caminhos que satisfazem as condições acima. Entretanto, só nos interessa o que minimize o custo da função:

$$DTW(R, V) = \left\{ \sqrt{\sum_{k=1}^{k} w_k} \right.$$

8. O caminho W pode ser facilmente encontrado pela programação dinâmica, já que o algoritmo consegue encontrar a próxima célula a avançar observando as adjacentes a ela, visando encontrar, como já dito, o caminho que minimize a função até chegar ao outro extremo da matriz. Assim, a distância cumulativa definida pela programação dinâmica $\gamma(i, j)$ e o mínimo das distâncias cumulativas das células adjacentes podem ser expressos como:

$$\gamma(i, j) = d(r_i, v_j) + min\left\{\gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1)\right\}$$

9. Ao fim do trajeto, o elemento da matriz, associado a $w_k = (n, m)$, ou seja, o último elemento do caminho percorrido, representa o valor cumulativo do caminho que define o processo de alinhamento entre teste e treino. Assim, esse valor é o coeficiente final de alinhamento.

3.2.12 Método EMD- *Earth Mover's Distace*

Comparado aos demais métodos deste trabalho, esse é, sem duvidas, o mais recente deles. Tem por finalidade assumir duas distribuições de características, uma de referência (treino) X e outra de teste Y , assim como em DTW, e calcular o quão distante uma está de se igualar à outra. O método tem como ideia principal associar a distribuição de treino a uma massa de "terra" adequadamente espalhada no espaço e a distribuição de

teste a um aglomerado de lacunas espalhadas no mesmo plano e espaço. O trabalho gasto para transportar "terra" da distribuição de treino a de teste, a fim de que a segunda se iguale à primeira, denomina a distância que há entre elas, o que exemplifica o nome EMD (RUBNER; TOMASI; GUIBAS, 2000).

Esse método adota também a ideia de associar pesos aos pontos, que não precisam ser próprios e não necessariamente iguais aos demais que compõem a distribuição. O peso u de um ponto $y \in Y$ está relacionado ao tamanho da lacuna em uma posição y do espaço, assim como um peso w de um ponto do conjunto X está relacionado à quantidade de "terra" existente naquele ponto. Cada elemento ou ponto de uma distribuição pode ser referenciado por meio de um ou mais valores que lhes servem como coordenadas, definindo sua posição no espaço.

É denominado o fluxo f_{ij} a quantidade de "terra" movida de um i-ésimo ponto para um j-ésimo. $F = f_ij \in F(X, Y)$ é o conjunto de todas as possibilidades de fluxo entre X e Y.

A seguir as restrições que definem o método EMD:

1- $f_{ij} \geq 0 \rightarrow 1 \leq i \leq m$ e $1 \leq j \leq n$;

2- $\sum_{j=1}^{n} f_{ij} \leq w_i \rightarrow \leq i \leq m$;

3- $\sum_{i=1}^{n} f_{ij} \leq u_i \rightarrow 1 \leq j \leq n$;

4- $\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} = min\{W, U\}$.

A restrição (1) garante que não haja quantidades negativas de "terra"; (2) e (3) garantem que cada lacuna não receba mais "terra" do que suporta e que não se retire mais do que existe em um determinado local; já a restrição (4) diz que, ao final do processo, todas as lacunas terão que ser preenchidas ou toda a "terra" utilizada.

3.3 Detalhes das implementações dos métodos estudados

A fase de implementação consistiu em transcrever para linguagem computacional a teoria aprendida durante o levantamento das técnicas estatísticas. Uma das principais metas deste trabalho é fazer com que os algoritmos retratem com afinco os conceitos levantados. Por isso, é de suma importância que os conceitos dos teoremas estejam bem entendidos.

3.3.1 *Z-Score*

A implementação do *z-score* foi bastante simples, visto que sua teoria não é muito extensa, assim como sua complexidade computacional. O método implementado possui

como parâmetro de entrada o *score* bruto, pelo qual é calculado a média aritmética e o desvio padrão. O retorno é obtido pelo cálculo da fórmula com esses três dados.

A implementação em método no Matlab apresentou um tamanho consideravelmente pequeno, tendo em vista que o programa aceita dados n-dimensionais, se comparado com linguagens populares, como C/C++, Java, entre outras. Tal fato se dá por esse software considerar todo dado como sendo uma matriz e por operações como soma e divisão de matrizes serem feitas diretamente, índice a índice, sem necessidade de criação de laços.

O cálculo da média aritmética em Matlab ocorre da seguinte forma: Se a entrada da função possui somente uma linha, a média é calculada normalmente, soma-se os elementos e divide pela quantidade. Se a entrada possui mais de uma linha, é calculada a média de cada coluna, e o retorno é um vetor de médias com o tamanho igual ao número de colunas da matriz de entrada. A função *mean* do Matlab pode ter como parâmetro de entrada, além da matriz de dados, o parâmetro numérico de valor 1 ou 2, que define o comportamento da função. O número 1 garante que a média seja calculada para cada coluna, já o 2 que seja calculada no sentido das linhas, assim, o retorno será um vetor com as medias de cada linha.

A seguir, pode-se observar a função *z-score* implementada na linguagem nativa do Matlab:

```
1 function z= zscore(a)
2     m = mean(a,1);
3     s = std(a,1);
4     N = size(a,1);
5     z = (a - repmat(m,[N,1])) ./ repmat(s,[N,1]);
```

Código 3.1 – Código fonte do método *Z-Score* implementado em Matlab

Note que, em Matlab, as funções de média e desvio padrão já se encontram implementadas. Em caso de matrizes, pode-se até calcular a média e o desvio das colunas, diferentemente da linguagem C, em que essas funções não se apresentam implementadas.

Ao se utilizar linguagens como C, faz-se necessário a utilização de iterações com for, quando se deseja fazer certa operação em todos os elementos de uma matriz. Em Matlab operações podem ser feitas diretamente pela função *repmat*, que replica as matrizes para tamanhos em comum, pois, como já dito, ele trabalha muito bem com matrizes e, por isso, consegue fazer esse tipo de cálculo sem que seja necessário o uso de iterações. Essa funcionalidade pode ser vista no código fonte do *z-score* em Matlab, descrito anteriormente. Observe o arquivo fonte a seguir, que ilustra a mesma implementação *z-score*, porém em linguagem C:

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

double media(double a[], int tam)
{
    double cont= 0;
    for (int i = 0; i < tam; i++)
    {
        cont+= a[i];
    }
    return cont/tam;
}

double desvioPadrao(double b[], int tam)
{
    double cont= 0;
    double media= media(b, tam);
    for(int i= 0; i> tam; i++)
    {
        cont += pow((b[i] - media), 2);
    }
    return sqrt(cont/tam);
}

int main()
{
    double m[]= {1,2,3,4,5};
    double d[5];
    double media= media(m, 5);
    double desvioP= desvioPadrao(m, 5);

    for(int i= 0; i< 5; i++){
        d[i]= (m[i] - media)/devioP;
    }
    system("PAUSE");
    return 0;
}
```

Código 3.2 – Código fonte do método *Z-Score*, implementado em linguagem C

3.3.2 Classificador KNN- *K Nearest Neighbours*

O método foi implementado para receber, como parâmetros de entrada, os seguintes dados: um inteiro representando o valor k, uma matriz de dados para treinamento, um vetor coluna contendo as classes da matriz de treinamento e a matriz de dados para teste que será classificada. A seguir, pode-se observar o código fonte do programa em Matlab:

```
function cl= knn(k, dadoscl, classes, dadosts, tipo)
    classes= classes(:);
    % metricas de distancia
    %     1= euclidiana
    %     2= cityblocks
    %     3= chessboard

    [Ac,Bc]= size(dadoscl);
    [At,Bt]= size(dadosts);
    cl= zeros(At,1);
    maior= max(classes);
    if k > Ac
        error('K deve ser um valor menor ou igual ao
        numero de linhas da matriz classificada');
    end
    if nargin == 4||tipo == 1
       aux= euclidiana(dadoscl, dadosts);
    else
       if tipo == 2
            aux= cityblocks(dadoscl, dadosts);
       end;
       if tipo == 3
            aux= chessboard(dadoscl, dadosts);
       end
    end
    for fg= 1:At
        v= aux(fg,:);
        [v2Temp, vIndTemp]= sort(v);
        temp= 0;
        n=0;
        while temp==0
            vInd= vIndTemp(1, 1:k - n);
            classesaux= classes(vInd,1);
            [RPT,cod]= repetemais2(classesaux,maior);
            if cod == 1
                temp = 1;
            else
                n= n+1;
            end
        end
        cl(fg, 1)= RPT;
    end

function [rp, cod]= repetemais2(v,m)
    vet = 1:m;
    a= hist(v,vet);
    [J, K]= sort(a(:), 1, 'descend');
```

```
48      rp= vet(K(1));
49      if J(1)> J(2)
50          cod= 1;
51      else
52          cod= 0; %codigo de erro em caso de empate
53      end
54
55  function aux= euclidiana (dadoscl, dadosts)
56      [Ac,Bc]= size(dadoscl);
57      [At,Bt]= size(dadosts);
58      aux= zeros(At,Ac);
59      for x= 1:At
60          for y= 1:Ac
61              aux(x,y)= sqrt(sum((dadosts(x,:)- dadoscl(y,:)).^2));
62          end
63      end
64
65  function aux= cityblocks (dadoscl, dadosts)
66      [Ac,Bc]= size(dadoscl);
67      [At,Bt]= size(dadosts);
68      aux= zeros(At,Ac);
69      for x= 1:At
70          for y= 1:Ac
71              aux(x,y)= sum(abs(dadosts(x,:)- dadoscl(y,:)));
72          end
73      end
74
75  function aux= chessboard
76      (dadoscl, dadosts)
77      [Ac,Bc]= size(dadoscl);
78      [At,Bt]= size(dadosts);
79      aux= zeros(At,Ac);
80      for x= 1:At
81          for y= 1:Ac
82             aux(x,y)= max(abs(dadosts(x,:)- dadoscl(y,:)));
83          end
84      end
85  end
```

Código 3.3 – Código fonte do método KNN

O programa foi desenvolvido da seguinte maneira: primeiramente, é realizado um teste para se certificar que o valor k inserido é valido, ou seja, se é menor ou igual ao número de linhas da matriz de treinamento. Caso não seja, uma mensagem de erro notifica o usuário.

Com o valor de k válido, o programa irá calcular as distâncias de cada elemento y_i de teste para todos os elementos de treinamento x_j, fazendo iterações por meio de laço for. Assim que as iterações de um elemento de Y terminam, as distâncias já estão armazenadas e o elemento é, então, classificado antes que as iterações trabalharem o próximo. Logo, quando é classificado o último elemento de Y, o programa não precisa voltar para concluir tarefas pendentes, demonstrando, assim, maior eficiência.

Como já visto na seção de levantamento de métodos, a classe atribuída ao elemento não classificado é aquela que apresenta maior frequência entre os k elementos mais próximos dele. Esse mecanismo, inicialmente, apresentava menor eficiência, pois percorria o vetor em busca dos k elementos mais próximos, capturando o índice de suas classes. Em seguida, percorria também as outras classes, anotando qual apresentava maior ocorrência. Posteriormente, esse método foi melhorado com a utilização de funções auxiliares, como *hist* e *sort*, que já se apresentam implementados na base de funções do Matlab. Ao invés de se percorrer o vetor para encontrar as classes dos k elementos mais próximos, foi utilizado *sort* para ordenar o vetor de forma decrescente, assim, os k primeiros elementos do vetor ordenado são os k "vizinhos mais próximos". A própria função *sort* já emite, como segundo parâmetro, os índices dos elementos, logo, os k primeiros índices representam também os índices das classes desses elementos.

O Matlab consegue acessar índices diretamente sem necessidade de iterações, então, tendo em mãos os índices das classes desejadas, elas são facilmente armazenadas em um vetor, para descobrir qual apresenta maior frequência. Isso foi descoberto utilizando a função *hist*, que constrói um histograma numérico referente à ocorrência de valores e, assim, é possível demonstrar qual classe predomina no vetor. Essa classe é atribuída ao elemento y_i, se não houver repetição de maior frequência. Esse último processo também é simples, pois, ordenando o vetor de histograma de forma decrescente, supõe-se que o primeiro elemento é o maior e que representa a quantidade de ocorrências de uma certa classe. Se o elemento da segunda posição apresentar um valor igual ao do primeiro, está constatado que houve empate de ocorrências, então, o software irá refazer o processo, tomando, agora, o valor de $k - 1$.

A métrica para o cálculo da distância mais usada é a da distância Euclidiana, porém também são utilizadas outras, como a distância de *Cityblock* (*Manhatans*) e *Chessboard*. Essas métricas de distâncias, visto que não se apresentavam na base de funções do Matlab, tiveram de ser implementas para a utilização no método de KNN.

- **Distância Euclidiana**

  Dados dois pontos n-dimensionais, $X = (x_1, x_2, \ldots, x_n)$ e $Y = (y_1, y_2, \ldots, y_n)$, a distância Euclidiana é computada como:

$$dist(X, Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_n - y_n)^2}$$

O conceito da distância Euclidiana é a raiz quadrada do somatório dos quadrados das diferenças de cada par de pontos de mesma ordem.

- **Distância *Cityblock* (*Manhatans*)**

  Essa métrica é definida como a somatória dos módulos das diferenças e possui a seguinte fórmula:

$$dist(X,Y) = |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_n - y_n|$$

- **Distância *Chessboard***

  Seu conceito é o valor máximo dos módulos das diferenças dos pontos em respectivas posições, cuja fórmula é:

$$dist(X,Y) = max(|x_1 - y_1|, |x_2 - y_2|, \ldots, |x_n - y_n|)$$

Para uma melhor visualização da diferença dessas três métricas, imaginemos uma matriz de dimensões 11 x 11, cujos elementos são zeros, com exceção do elemento central, que possui valor 1, como mostrado na Figura 6.

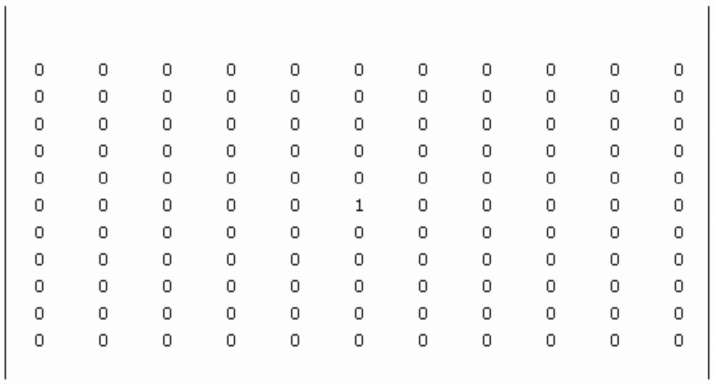

Figura 6 – Matriz referência para aplicação de medidas

Existe uma função chamada *bwdist*, do Matlab, que calcula a distância de todos os zeros de uma matriz para o elemento 1 mais próximo. Como só existe um e ele representa o centro, a função retratará perfeitamente a distância de cada elemento até o centro da matriz, mostrando com uniformidade a diferença das métricas citadas. As figuras 7, 8 e 9 mostram, respectivamente, como é dado o resultado, utilizando cada uma métricas Euclidiana, *Citybock* e *Chessboard*:

3.3.3 Método discriminante de Fisher

Inicialmente, o método foi implementado para abranger um campo de somente duas classes. Isso ajudou no aprendizado gradual do teorema e fez com que a implementação para varias classes ficasse mais fácil.

Na versão do método de discriminação para duas classes, os parâmetros de entrada são um vetor coluna contendo as classes, a matriz de dados classificados com elas e uma segunda matriz com os dados não classificados. A saída é um vetor coluna contendo as novas classes, em que a dimensão das linhas é igual à da matriz.

O método implementado é específico para duas populações, portanto, uma entrada de parâmetro com dados que apresentem mais de duas classes são devidamente detectados e uma mensagem de erro é mostrada na tela.

Os parâmetros de entrada são três: o vetor coluna, contendo as classes do treino; a matriz de dados de treino; e a matriz de teste que se deseja classificar.

A implementação utilizou apenas funções já contidas na base do Matlab, como:

- *inv* : que devolve a inversa de uma matriz;
- $find$: que procura elementos com determinada característica;
- *ones* : que aloca uma matriz com todos elementos iguais a um;
- *eig* : responsável por calcular os autovetores de uma matriz.

A implementação do método de Fisher para várias classes apresentou diversas facilidades, pois uma parte dele pôde ser aproveitada da implementação do método que funcionava para o de duas classes.

Inicialmente, é realizado um teste de compatibilidade para verificar se o número de variáveis em cada amostra dos dois conjuntos X e Y, treino e teste, respectivamente, são correspondentes. Caso não sejam, não faz sentido continuar a execução. Caso contrário, um novo teste será feito para verificar se o tamanho do vetor de classes é igual à quantidade de amostras no conjunto X. Assim como no outro teste, em caso de correspondência, o programa continuará a execução. Esse teste é padrão para todos os classificadores, a fim de evitar erros de compilação e diminuir o custo computacional.

Assim como no método especifico para duas classes, esse também está preparado para receber os seguintes parâmetros de entrada: o vetor coluna, contendo as classes do treino; a matriz de dados de treino; e a matriz de teste a qual se deseja classificar. A implementação utilizou funções já presentes na base de funções do Matlab, visando otimização.

Com isso, os métodos de classificação de Fisher ficam completos, podendo ser utilizados tanto para duas classes quanto para várias. Abaixo, a representação de como código fonte desse método é implementado:

```
1 function f= fisher(classes, pop,v)
```

```
    if (size(pop,2) ~= size(v,2))
        error('Numero de variaveis do teste nao compativel com o de
    treinamento');
    end
    numClass= max(classes);
    minn= min(numClass - 1, size(pop,2));
    Col= size(pop, 2);
    B= zeros(Col, Col);
    W= B;
    P2= zeros(1, minn);
    P3= zeros(1, numClass);
    tampopN= zeros(1, numClass);
    newClass= zeros(size(v,1), 1);
    for N= 1:numClass
        popN= pop(find(classes == N),:);
        tampopN(N)= size(popN,1);
        xl(:,N)= mean(popN,1)';
        S{N}= cov(popN);
    end
    mi= mean(xl,2);
    for N= 1:numClass
        d1= (xl(:,N) - mi);
        d2= d1';
        B=(B +(d1 * d2));
        W= W + ((tampopN(N) - 1) * S{N});
    end
    Sp= (1/(sum(tampopN)- numClass))* W;
    q= pinv(Sp')* B;
    [autovetores, ~] = eig(q);
    for M= 1:size(v,1)
        for N= 1:numClass
            for O= 1:minn
                autoVec= autovetores(:,O);
                autoVec= autoVec(:)';
                x= v(M,:)';
                media= xl(:,N);
                media= media(:);
                e1= (x - media);
                e2= (autoVec * e1)^2;
                P2(O)= e2;
            end
             P3(N)= sum(P2);
        end
        [CC, newClass(M)]= min(P3);
    end
    f= newClass(:);
```

Código 3.4 – Código fonte do método de Fisher

3.3.4 Método discriminante de Bayes

Dos métodos implementados, esse foi o que impôs maiores dificuldades. O método de Bayes é um classificador que determina as classes das amostras, baseado em dados já classificados. Tome um conjunto Y de dados não classificados e que se deseja classificar, baseado em um outro conjunto X, porém que já possui classificação. Esse processo é feito tendo como base a equação da probabilidade a posteriori, que equivale ao produto da densidade probabilidade e a probabilidade a priori:

$$P(C|Y) = P(Y|C)P(C)$$

Sabendo que os conjuntos X e Y são parecidos, é de se esperar que o resultado da classificação de Y seja igual à de X, ou pelo menos similar. Essa foi a forma de teste utilizada para provar a eficiência dos métodos implementados, tanto os anteriores quanto o de classificação bayesiana. Porém, não foi esse o resultado obtido, e constatou-se que o erro estava na obtenção da probabilidade a priori de cada classe, que havia sido calculada de maneira errada.

Tome como exemplo a matriz de amostras X de cinco linhas, ou seja, cinco amostras de dados classificados em duas classes, 1 e 2, cujo vetor coluna C mostra as classes as quais cada conjunto de dados amostrais (linhas) pertencem, observe a Figura 10.

A probabilidade a priori de uma nova amostra não classificada pertencer a certa classe se dá pela quantidade de elementos nessa classe em relação ao total de elementos do conjunto X. Por exemplo, há três amostras classificadas como classe 1, então, sua probabilidade a priori de ocorrer é de 3/5, já que existem cinco amostras no total, enquanto a probabilidade da classe 2 ocorrer é de 2/5.

Note que essa probabilidade é dada antes mesmo de se conhecer os dados a serem classificados, e que, segundo o teorema de Bayes, ela é multiplicada na fórmula, então, uma classe com uma probabilidade maior tem sempre mais chance de se atribuir a dados não classificados.

Depois de resolvido o problema da probabilidade a priori, foram realizados novos testes, porém novos distúrbios foram detectados nos resultados. Logo, foi constatado que o problema era um mau condicionamento de dados, pois a matriz inversa da covariância apresentava, às vezes, valores muito pequenos (próximos de zero) ou muito grandes. Foi construído, então, um novo programa para resolver o problema. Ele teve como base um caso especial de Bayes, que supõe que, na matriz de covariâncias, os dados de uma amostra são estatisticamente independentes. Nesse caso, pode-se considerar apenas a diagonal principal da matriz. A seguir, pode-se observar o código fonte do programa implementado:

```
function newClass= bayesCasoEspecial(classes, pop, v, t, varargin)
   [C,D]= size(v);
   [A,B]= size(pop);
   if(D ~= B)
       error('O numero de variaveis por amostra nao eh equivalente');
   end
   if nargin == 4
       if t == 1
           %funcao de densidade probabilidade ou verossimilianca usa
  distribuicao normal%
           funcDp= 1;
       elseif t == 2
           %funcao de densidade probabilidade ou verossimilianca usa
  distribuicao lognormal
           funcDp= 2;
       else
           erro('para quatro parametros somente opcao 1 ou 2');
       end
   end
   if t== 3
       if nargin == 6
           shapee= varargin{1};
           scalee= varargin{2};
           %funcao de densidade probabilidade ou verossimilianca usa
  distribuicao gamma
           funcDp= 3;
       else
           %em caso de somente uma entrada o parametro de escala eh
  considerado 1
           shapee= varargin{1};
           %distribuicao gamma com um soh parametro
           funcDp= 31;
       end
   end
   if nargin == 5 && t==4
       lambdaa= varargin{1};
       %funcao de densidade probabilidade ou verossimilianca usa
  distribuicao poisson
       funcDp= 4;
   elseif (nargin ~= 5 && t==4)
       error('opcao de funcao de densidade
       probabilidade ou numero de entradas invalidas');
   end
   numClass= max(classes);
   newClass= zeros(C,1);
   medias= zeros(numClass,B);
   sigma= zeros(numClass,B);
```

```
43      pp= zeros(1,numClass);
44      for N= 1:numClass
45          popN= pop(find(classes == N),:);
46          pp(N)= size(popN,1)/A;
47          medias(N,:)= mean(popN,1) ;
48          sigma(N,:)= std(popN,0,1);
49      end
50      for M= 1:C
51          x= v(M,:);
52          gs= zeros(1,numClass);
53          if funcDp == 1
54              for N= 1:numClass
55                  pptemp= normpdf(x,medias(N,:),sigma(N,:));
56                  gs(N)= prod(pptemp)*pp(N);
57              end
58          end
59          if funcDp == 2
60              for N= 1:numClass
61                  pptemp= lognpdf(x,medias(N,:),sigma(N,:));
62                  gs(N)= prod(pptemp)*pp(N);
63              end
64          end
65          if funcDp == 3
66              for N= 1:numClass
67                  pptemp= gampdf(x, shapee, scalee);
68                  gs(N)= prod(pptemp)*pp(N);
69              end
70          end
71          if funcDp == 31
72              for N= 1:numClass
73                  pptemp= gampdf(x,shapee);
74                  gs(N)= prod(pptemp)*pp(N);
75              end
76          end
77          if funcDp == 4
78             for N= 1:numClass
79                  pptemp= poisspdf(x,lambdaa);
80                  gs(N)= prod(pptemp)*pp(N);
81             end
82          end
83         [a,newClass(M)]= max(gs);
84      end
85  end
```

Código 3.5 – Código fonte do método de Bayes

Como esse método considera que os dados são independentes, a média dos dados de uma população é obtida separadamente. Assim, se um conjunto possui G elementos,

e eles possuem H variáveis, existirá também um número H de médias que representam essa população, pois uma variável y_{i1} de um elemento se relaciona com toda variável y_{j1} dos outros elementos para cálculo da média. Esse tipo de operação independente também é utilizado para a obtenção da matriz de covariâncias. Depois de obtidas essas variáveis, é possível efetuar o cálculo da distribuição normal.

Porém, como há H variáveis, existirão também H resultados da equação para cada população, assim, é fácil obter um valor único para a probabilidade, calculando o produtório desses valores.

$$P(Y|C) = \sum_{i=1}^{H} p_i$$

em que p_i é o valor encontrado por meio da equação da distribuição normal, utilizada com cada uma das H variáveis. Por fim, é o resultado do produtório, que é multiplicado como da probabilidade a priori de cada classe.

Partindo para outras formas, chegou-se à conclusão que, voltando a utilizar o programa anterior, era possível utilizar a função *std* (*standard desviation*, ou desvio padrão), substituindo a *cov*, que gera a matriz de covariâncias, também implementada no Matlab.

Após mais alguns dias de estudo, notou-se que, na forma de resolver o problema, no momento do cálculo do *std* (desvio padrão) para criação da matriz de covariância, era necessário acrescentar um parâmetro de entrada que representasse o $flag$ igual a 0, ou seja, assim, a função calcularia o desvio padrão ao longo da dimensão. Isso elimina os resultados negativos da matriz, chegando a um resultado dentro do esperado para o cálculo. Além dessas alterações, foi utilizado uma função descoberta, a *normpdf*, que já existia e era implementada no Matlab, que substituiu bem a fórmula para resolução da densidade probabilidade, equação da distribuição normal, já explicada na seção de levantamentos.

Com esse caso especial já funcionando, o próximo passo foi tentar corrigir o programa anteriormente construído. Por meio de pesquisas, constatou-se que o problema de mau condicionamento de dados era devido à forma como o cálculo da matriz inversa era realizado pela função *inv* do Matlab. Esse cálculo foi substituído por uma outra função de cálculo da inversa de uma matriz, utilizando a decomposição LU e pseudo-determinantes, como é explicado a seguir.

3.3.4.1 Decomposição LU

Do inglês *lower* e *upper*, esse método geralmente é usado nos casos em que se deseja resolver vários sistemas lineares em que a matriz de coeficientes é a mesma (A). O método, então, visa decompor a matriz para resolver o sistema linear $Ax = b$.

Por ele, então, a matriz A é decomposta como resultante de um produto de duas matrizes L e U, em que L representa uma matriz triangular inferior e U representa uma matriz triangular superior. Logo, $A = L.U$.

Tendo como base as informações acima, pode-se, então, reescrever o sistema $Ax = b$ como $Ax = (LU)x = L(Ux) = b$.

Fazendo-se $Ux = y$, podemos resolver o sistema $Ax = b$ em duas etapas:

1. Resolve-se o sistema triangular inferior $L_y = b$, tendo como solução $\bar{y}$;
2. Substitui-se o valor $L_y = b$ obtido no sistema $Ux = \bar{y}$, chegando à nova solução $\bar{x}$.

Em outras palavras, a resolução de um sistema linear $Ax = b$ pode ser decomposta pela de dois sistemas lineares ($L_y = b$ e $U_x = y$).

A aplicação do método LU possui uma condição: que o determinante da matriz A seja diferente de zero. Contudo, não é usual checá-la, visto que calcular o determinante é mais trabalhoso do que o próprio método. A decomposição LU, inclusive, é inviável em caso de alguma divisão por zero durante seu processo.

3.3.4.2 Pseudo-determinante

Em estatística, o pseudo-determinante é o produto de todos os autovalores de uma matriz quadrada diferentes de zero. É indicado para os casos em que a matriz é singular, ou seja, não é inversível, do contrário, ele coincide com o determinante usual.

O pseudo-determinante de uma matriz quadrada A, de dimensões N por N, pode ser definido como:

$$|A|_+ = lim_{\alpha \to 0} \frac{|A + \alpha I|}{a^{n-rank(A)}}$$

em que I representa a matriz identidade de A e $rank(A)$ representa a ordem.

Se A é tomada como uma matriz positiva, então, os valores singulares e autovalores de A se equivalem. Nesse caso, se a decomposição em valores singulares (SVD) está disponível, pode-se calcular o produto dos valores singulares não-nulos. Se todos os valores singulares forem iguais a zero, então, o pseudo-determinante é 1.

O pseudo-determinante é usualmente empregado em saídas de softwares estatísticos, quando a matriz de covariâncias é singular, propondo uma forma de inversão, caso seja preciso. Porém, há casos em que nem LU e nem pseudo-determinantes funcionam, por haver dados muito ruins nas amostras. Assim, é necessário a utilização de seleção de características, um método ainda não abordado pelo projeto.

3.3.5 Método discriminante *Nearest Prototype*

O algoritmo possui como entrada quatro parâmetros: a matriz de dados de treino; seu respectivo vetor de classes; a matriz de testes que deseja-se classificar; e o tipo que discrimina qual distância será usada para o calculo (Euclidiana, *Cityblocks* ou *Chessboard*).

Após identificar o tipo de métrica de distância a ser utilizada, o programa realiza dois laços de processamento, um dentro do outro. O mais externo toma os dados de teste e prepara as matrizes auxiliares, que são criadas especificamente para o algoritmo e servem para armazenar dados a fim de utilizá-los mais tarde, economizando, assim, processamento de alocação de memória durante o laço. O mais interno calcula a equação de protótipos para cada classe. Entre o final de uma iteração do laço interno e o do externo, ou seja, antes de se descartar um elemento de teste, verifica-se qual o protótipo de maior valor e atribui a ele a classe referenteções vão se repetir até que toda a matriz de teste esteja classificada.

A seguir, pode-se observar o algoritmo referente a esse método:

```
function cl= prototype(dadoscl, classes, dadosts, tipo)
    classes= classes(:);
    [Ac,Acd]= size(dadoscl);
    [At,~]= size(dadosts);
    cl= zeros(At,1);
    maior= max(classes);
    mediaClass= zeros(maior, Acd);
    m= 2;
    for Q= 1:maior
        mediaClass(Q,:)= mean(dadoscl(find(classes == Q),:));
    end
    if nargin == 4||tipo == 1
        func= @euclidiana;
    else
       if tipo == 2
            func= @cityblocks;
       end
       if tipo == 3
            func= @chessboard;
       end
    end
    mi= zeros(maior, 1);
    for fg= 1:At
        lk= dadosts(fg,:);
        sZ= zeros(1,maior);
        dDist= zeros(1,maior);
        for y= 1:maior
```

```
28              mCl= mediaClass(y,:);
29              dDist(y)= func(lk, mCl);
30              sZ(y)= 1 /((dDist(y)).^(2/(m-1)));
31          end
32          sSumDist= sum(sZ);
33          mi= sZ./sSumDist;
34         [~, cl(fg, 1)]=  max(mi);
35      end
36
37      function aux= euclidiana (dadoscl, dadosts)
38          [Ac,~]= size(dadoscl);
39          [At,~]= size(dadosts);
40          aux= zeros(At,Ac);
41          for x= 1:At
42              for y= 1:Ac
43                  aux(x,y)= sqrt(sum((dadosts(x,:)- dadoscl(y,:)).^2));
44              end
45          end
46
47      function aux= cityblocks (dadoscl, dadosts)
48          [Ac,~]= size(dadoscl);
49          [At,~]= size(dadosts);
50          aux= zeros(At,Ac);
51          for x= 1:At
52              for y= 1:Ac
53                  aux(x,y)= sum(abs(dadosts(x,:)- dadoscl(y,:)));
54              end
55          end
56
57      function aux= chessboard(dadoscl, dadosts)
58          [Ac,~]= size(dadoscl);
59          [At,~]= size(dadosts);
60          aux= zeros(At,Ac);
61          for x= 1:At
62              for y= 1:Ac
63                 aux(x,y)= max(abs(dadosts(x,:)- dadoscl(y,:)));
64              end
65          end
66  end
```

Código 3.6 – Código fonte do método *Nearest Prototype*

3.3.6 Método discriminante NFL- *Nearest Feature Line*

No algoritmo, são necessárias apenas três entradas de parâmetros: a matriz de amostras de treino classificada, seu respectivo vetor de classes, além dos dados de teste. É feita a verificação desses dados para a continuação do programa.

É necessário saber previamente quantas serão as retas (FL) formadas por questão de otimização, pois, a partir desse valor, é possível fazer uma pré-alocação de memória. Além disso, permite-se prever esse valor por meio de uma combinação da quantidade de amostras dentro de uma classe, tomadas duas a duas, ou seja, $C_{n,2}$. Sabendo que existem g classes, essa combinação será feita g vezes. Esse resultado representa a quantidade de retas que serão alocadas posteriormente na memória. Essa operação é resolvida por meio de um laço computacional.

Para cada uma das retas, é calculada a distância euclidiana entre ela e a amostra de teste y_i, anteriormente descritos neste relatório. Ao final, descobre-se qual a reta que possui menor distância ao ponto, utilizando a função $min()$. A classe referente aos pontos que compõem essa reta será associada à amostra de teste. Esse processo é aplicado a todos os elementos do conjunto de teste Y. A seguir, observe o métodos NFL implementado em Matlab:

```
function f= nfl(dados, classes, v)
    [A, B]= size(dados);
    [C, D]= size(v);
    E= length(classes);
    newClass= zeros(1,C);
    if(B ~= D)
        error('Numero de colunas de teste e de amostra incompativeis');
    end
    if( A ~= E)
        error('Quantidade de classes e de dados incompativeis');
    end
    numClass= max(classes);
    tam= 0;
    ct2= 1;
    for x= 1:numClass
        tam= (tam + nchoosek((length(find(classes==x))),2));
    end
    retas= zeros(tam,2);
    for s= 1: A-1
        a= find(classes==classes(s));
        b= a(find(a>s));
        lb= length(b);
        if (~isempty(b))
            retas(ct2:ct2+lb-1,1)= repmat(s,lb,1);
            retas(ct2:ct2+lb-1,2)= b;
            ct2= ct2+ lb;
        end
    end
    d= zeros(tam, 1);
    classTemp= zeros(tam,1);
```

```
    for r= 1: C
        xq= v(r,:)';
        for s= 1:tam
            xi= dados(retas(s,1),:)';
            xj= dados(retas(s,2),:)';
            mi= ((xq - xi)' * (xj - xi))/ ((xj - xi)' * (xj - xi));
            xij= ((1 - mi) * xi) + mi* xj;
            d(s)= euclidiana(xq,xij);
            classTemp(s)= classes(retas(s,1));
        end
        [~,ind]= min(d);
        newClass(r)= classTemp(ind);
    end
    f= newClass;
    function h= euclidiana(u,y)
       h= sqrt(sum((u-y).^2));
    end
end
```

Código 3.7 – Código fonte do método NFL

3.3.7 Método discriminante SFL- *Shortest Feature Line Segment*

Assim como NFL, esse método possui como entrada três parâmetros: a matriz de amostras de treino classificada, seu respectivo vetor de classes, além dos dados de teste. É feito também a verificação desses dados para a continuação do programa. A tática de previsão do tamanho da matriz de FLs é a mesma utilizada no método NFL, por meio de combinação matemática, visando a economia de memória por menor alocação.

Realiza-se uma sequência de cálculos de coeficientes α e a atribuição de valor infinito à distância dos pontos x_i^θ e x_j^θ que compõe cada FL com α com valor menor que 90. Assim, a função *min*, que verifica o menor, nunca vai escolher esse valor para associar a amostra de teste. O resultado dessa função define em qual classe o elemento deve ser alocado, como descrito em 3.2.10. Essa é a diferença que existe entre esse método e o NFL, e que pode ser observada no código fonte que segue.

```
function f= sfl(dados, classes, v)
    [A, B]= size(dados);
    [C, D]= size(v);
    E= length(classes);
    newClass= zeros(1,C);
    if(B ~= D)
        error('Numero de colunas de teste e de amostra incompativeis');
    end
    if( A ~= E)
```

```
10          error('Quantidade de classes e de dados incompativeis');
11      end
12      numClass= max(classes);
13      tam= 0;
14      ct2= 1;
15      for x= 1:numClass
16          tam= (tam + nchoosek((length(find(classes==x))),2));
17      end
18      retas= zeros(tam,2);
19      for s= 1: A-1
20          a= find(classes==classes(s));
21          b= a(find(a>s));
22          lb= length(b);
23          if (~isempty(b))
24              retas(ct2:ct2+lb-1,1)= repmat(s,lb,1);
25              retas(ct2:ct2+lb-1,2)= b;
26              ct2= ct2+ lb;
27          end
28      end
29      d= zeros(tam, 1);
30      classTemp= zeros(tam,1);
31      for r= 1: C
32          xq= v(r,:)';
33          for s= 1:tam
34              xi= dados(retas(s,1),:)';
35              xj= dados(retas(s,2),:)';
36              alpha= ((180/pi) * acos(((xq - xi)'*(xq - xj))/((euclidiana(xq,
    xi))*(euclidiana(xq,xj)))));
37              if(alpha<90)
38                d(s)= Inf;
39                classTemp(s)= Inf;
40              else
41                d(s)= euclidiana(xi,xj);
42                classTemp(s)= classes(retas(s,1));
43              end
44          end;
45          [~,ind]= min(d);
46          if(classTemp(ind)~=Inf)
47          newClass(r)= classTemp(ind);
48          end
49      end
50      UnClassInds= find(newClass==0);
51      if (~isempty(UnClassInds))
52          k= length(dados);
53          newClass(UnClassInds)= knn(k, dados, classes,v(UnClassInds), 1);
54      end
55      f= newClass;
```

```
56      function h= euclidiana(u,y)
57          h= sqrt(sum((u-y).^2));
58      end
59  end
```

Código 3.8 – Código fonte do método SFL

3.3.8 Método discriminante DTW- *Dynamic Time Warping*

O programa é bastante simples, possui apenas dois parâmetros de entrada: sequências de dados considerada como treino e que servirá como referência, além dos dados de teste. O intuito do algoritmo é calcular o coeficiente que represente a diferença entre as duas sequências, ou seja, o quão distante está a segunda de se equiparar à primeira. Inicialmente, a operação do algoritmo é pré-alocar a matriz de distâncias acumuladas, prevendo o tamanho final dela por meio da função *repmat* e depois preenchê-la. Esse método permite que a alocação seja feita somente uma vez, diminuindo o custo computacional. Essa prática pode não parecer tão vantajosa em amostras pequenas, porém, em amostras de grande escala, como o custo computacional, é reduzido n x m vezes, em que n e m são as dimensões da matriz de distâncias acumuladas. Fica claro o quanto ele é mais eficiente. Após a alocação, o preenchimento é feito utilizando os passos descritos em 3.2.11. Primeiramente, a primeira linha e, posteriormente, a primeira coluna. Como Matlab é um software especialista em operações matriciais, o acesso aos índices para preenchimento é facilmente realizado. Da mesma forma, é possível realizar o preenchimento do restante da matriz, acessando as células adjacentes. A descoberta do caminho W também é realizado de forma simples. Uma célula da matriz que se quer preencher observa suas células adjacentes, sempre consecutivas, aplicando a equação descrita na seção de levantamento de bases estatísticas. Ao fim das iterações, o último elemento do caminho W contém o coeficiente acumulado e, portanto, final, que expressa a diferença que há entre a sequência de treino e teste. A seguir, pode-se observar a implementação desse método:

```
1  function [Dist,D,k,w]= dtw(t,r)
2      [~,N]=size(t);
3      [~,M]=size(r);
4      d=(repmat(t(:),1,M)-repmat(r(:)',N,1)).^2;
5      D=zeros(size(d));
6      D(1,1)=d(1,1);
7      for n=2:N
8          D(n,1)=d(n,1)+D(n-1,1);
9      end
10     for m=2:M
11         D(1,m)=d(1,m)+D(1,m-1);
12     end
```

```
13      for n=2:N
14          for m=2:M
15              D(n,m)=d(n,m)+min([D(n-1,m),D(n-1,m-1),D(n,m-1)]);
16          end
17      end
18      Dist=D(N,M);
19      n=N;
20      m=M;
21      k=1;
22      w=[];
23      w(1,:)=[N,M];
24      while ((n+m)~=2)
25          if (n-1)==0
26              m=m-1;
27          elseif (m-1)==0
28              n=n-1;
29          else
30              [~,number]=min([D(n-1,m),D(n,m-1),D(n-1,m-1)]);
31              switch number
32                  case 1
33                      n=n-1;
34                  case 2
35                      m=m-1;
36                  case 3
37                      n=n-1;
38                      m=m-1;
39              end
40          end
41          k=k+1;
42          w=cat(1,w,[n,m]);
43      end
44  end
```

Código 3.9 – Código fonte do método DTW

3.3.9 Método EMD- *Earth Mover's Distance*

O método EMD implementado no trabalho possui, como parâmetros de entrada, quatro sequências de valores. Os dois primeiros são as matrizes de características de treino e teste. A quantidade de linhas da matriz representa a quantidade de características da distribuição. As colunas representam as coordenadas daquela característica no plano, por exemplo, uma característica com três valores a referenciando possui coordenadas que dizem respeito aos eixos x, y e z no espaço. No entanto, poderia ser representada por mais ou menos valores que a definissem, por exemplo, no tempo, ou mesmo outras características que dependem muito do problema e do fim ao qual o método é utilizado. O importante a saber é que, independente da quantidade de valores referenciando uma

característica, esse valor deve ser o mesmo para as demais características da mesma e da outra distribuição, seja ela treino ou teste. Os dois outros parâmetros de entrada são cadeias de pesos referentes às características. Cada vetor ou cadeia de pesos deve ser do mesmo tamanho do número de linhas de sua matriz de características correspondentes. Além disso, cada peso do vetor é um valor único e referencia a quantidade de "terra" de um ponto ou tamanho da lacuna, dependendo de qual distribuição pertença.

Usualmente a matriz de treino é um parâmetro que precede a de teste, essa ordem também serve para seus respectivos pesos e a primeira possui tamanho maior. Porém a implementação do algoritmo permite que o usuário entre com a ordem contrária de dados, pois ele verifica os tamanhos das matrizes e, caso a primeira seja menor que a segunda, ele assume que esta é a matriz de características de treino, associando, assim, também seu vetor de pesos. Essa manobra é necessária, pois o processamento do método EMD em que a distribuição de teste possua maior número de características que a de treino é considerada errada e fere as condições de cálculo do método citadas na seção de levantamento do método. Tomando como base a idéia principal do EMD, que visa calcular o custo de transporte de "terra" do treino ao teste até que o segundo seja igual ao primeiro, isso seria impossível com um teste maior que o treino, pois, dessa forma, haveria um transporte de "terra" negativa ou de sentido contrário até que se igualassem.

O algoritmo implementado monta a matriz de distâncias acumuladas, DA, pré-alocada, preenchendo-a e calculando a distância euclidiana entre as características, preparando, assim, para os próximos cálculos.

O cálculo final é bastante simples, já que o Matlab possui uma função pronta que trata programação linear de maneira simplificada. Essa função recebe alguns dados como parâmetros de entrada: um vetor de custo que é a transformação da matriz DA em um vetor coluna, duas matrizes de restrições, além de um limite inferior para manter os resultados a cima deste. A função minimiza o resultado da equação com o mínimo de iterações. A seguir, a implementação deste método:

```
1  function dist= emd(c1, c2, p1, p2)
2      [M A] = size(c1);
3      [N B] = size(c2);
4      if(N>M)
5          aux= c1;
6          c1= c2;
7          c2= aux;
8          aux2= p1;
9          p1= p2;
10         p2= aux2;
11         aux3= M;
12         M= N;
```

```
13          N= aux3;
14      end
15      mDes1 = zeros(M, M*N);
16      mDes2 = zeros(N, M*N);
17      mCusto= zeros(M,N);
18      if A == B
19          if(M ~= length(p1) || N ~= length(p2))
20              error('A quantidade de pesos e de caracteristicas so
    imcompataveis');
21          end
22      else
23          error('numero de colunas nas caracteristicas sao imcompativeis');
24      end
25      for x= 1:M
26          for y= 1:N
27              mCusto(x,y)= euclidiana(c1(x,1:A), c2(y,1:A));
28          end
29      end
30      tt= mCusto';
31      vCusto= tt(:);
32      for x= 1:M
33          for y= 1:N
34              k= y+ (x- 1)* N;
35              mDes1(x,k)= 1;
36              mDes2(y,k)= 1;
37          end
38      end
39      mDesig= [mDes1; mDes2];
40      vDesig= [p1(:);p2(:)];
41      mIgual= ones(M+N, M*N);
42      vIgual= ones(M+N, 1)* min(sum(p1),sum(p2));
43      li= zeros(1, M*N);
44      [vec, zTemp]= linprog(vCusto, mDesig, vDesig, mIgual, vIgual,li);
45      dist= zTemp/ sum(vec);
46
47      function e= euclidiana(u,y)
48         e= sqrt(sum((u-y).^2));
49  end
```

Código 3.10 – Código fonte do método EMD

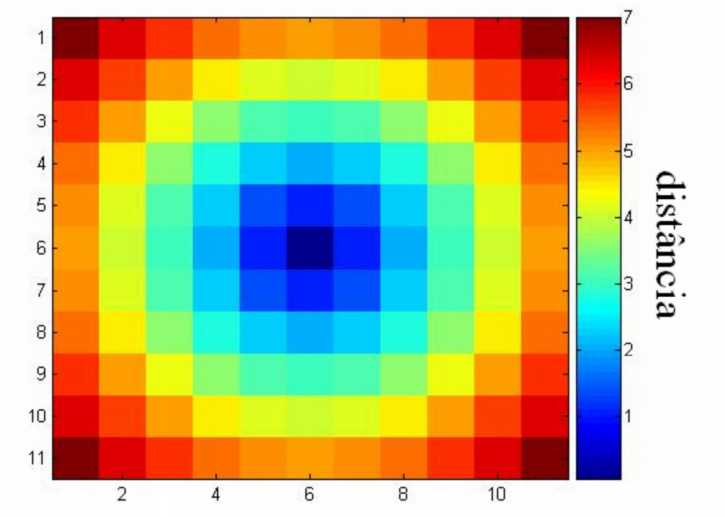

Figura 7 – Demostração da métrica Euclidiana aplicada à matriz referência

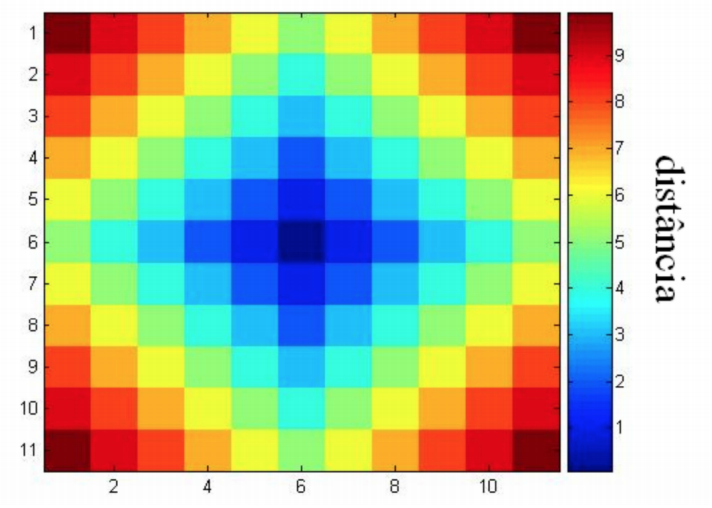

Figura 8 – Demostração da métrica *Cityblock* aplicada à matriz referência

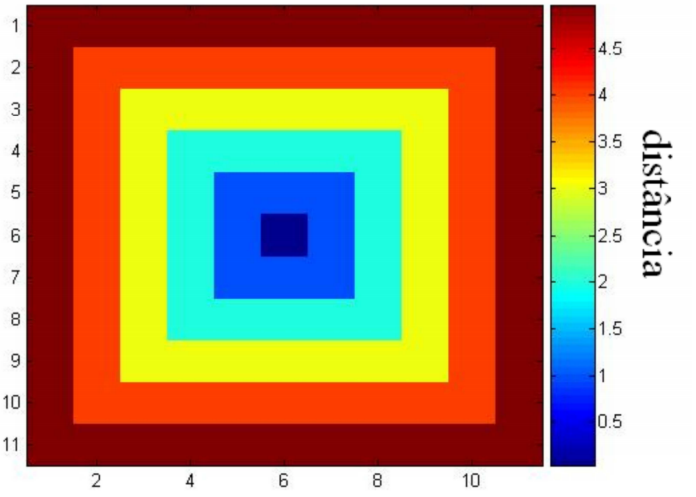

Figura 9 – Demostração da métrica *Chessboard* aplicada à matriz referência

C	X	
1	-0.4326	1.1909
1	-1.6656	1.1892
2	0.1253	-0.0376
2	0.2877	0.3273
1	-1.1465	0.1746

Figura 10 – Representação do vetor de classes C e a matriz de dados X.

Conclusão

A implementação da *toolbox* contendo os métodos estatísticos foi finalizada como esperado. Esses métodos poderão ser utilizados nas mais diversas áreas que necessitem de classificação e comparações em escalas multivariadas, proporcionando melhor eficácia e agilidade em processos realizados em diversos campos de conhecimento, por exemplo, tarefas de comparações de cadeias de DNA/RNA e padrões de fala ou mesmo classificações de dados para separação de populações para armazenamento em bancos de dados ou previsões estatísticas para amostragem etc.

Todas as tarefas previstas para o trabalho no cronograma foram concluídas com êxito, podendo destacar algumas, como a implementação de métodos bastante recentes, que, inclusive, ainda possuem poucas referências para estudo ou precedentes no que diz respeito à nossa proposta de trabalho. DTW, EMD, NFL e SFL são métodos que podem ser citados como exemplos disso. Não se pode deixar de citar a implementação de métodos mais antigos e que apresentam grande importância no contexto de classificação estatística como Fisher, Bayes, *Prototype* etc.

Concluindo, ao final das atividades de pesquisa e implementação, o resultado foi o esperado e a implementação desses métodos podem até vir a ser utilizados em outros projetos de pesquisa.

Referências

ANDERSON, T. W. *An Introduction to Multivariate Statistic Analysis*. New York: Wiley, 1958. Citado na página 13.

BEZDEK, J. C.; KUNCHEVA, L. Nearest prototype classifier designs: An experimental study. *Int. J. Intell. Syst*, v. 16, n. 12, p. 1445–1473, 2001. Citado 2 vezes nas páginas 16 e 28.

CHAMBERS, J. M. *Software for Data Analysis*. New York, NY 10013, USA: Springer, 2008. Citado na página 19.

DUDA, R. O.; HART, P. P. E.; STORK, D. G. *Pattern classification*. 2. ed. New York: Wiley, 2001. Citado 2 vezes nas páginas 16 e 29.

EATON, M. L. *Multivariate Statistics*. New York: John Wiley, 1983. Citado na página 14.

EVERITT, B. S.; DUNN, G. *Applied Multivariate Data Analysis*. 2. ed. [S.l.]: Arnold, 1991. Citado na página 14.

FANG, C. From dynamic time warping (dtw) to hidden markov model (hmm). 2009. Citado 2 vezes nas páginas 17 e 32.

HAIR, J. F. J. et al. *Análise Multivariada de Dados*. 5. ed. Porto Alegre: Bookman, 2005. Citado 2 vezes nas páginas 15 e 16.

HAN, D.-Q.; HAN, C.-Z.; YANG, Y. A novel classifier based on shortest feature line segment. *Pattern Recognition Letters*, v. 32, n. 3, p. 485–493, 2011. Citado 4 vezes nas páginas 7, 16, 29 e 31.

KELLER, J. M.; GRAY, M. R.; GIVENS, J. A. A fuzzy K-nearest neighbor algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 15, n. 4, p. 580–585, 1985. Citado na página 14.

MALAKOFF. A brief guide to bayes theorem. *SCIENCE: Science*, v. 286, 1999. Citado na página 15.

MOLINA, E. C. Bayes' theorem - an expository presentation. *Bell system Technical Journal*, p. 273–283, 1931. Citado 2 vezes nas páginas 15 e 24.

MORENO-SECO, F.; MICÓ, L.; ONCINA, J. *A fast approximately k-nearestneighbour search algorithm for classication*. 2000. OAI-PMH server at citeseerx.ist.psu.edu. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.2931>. Acesso em: 24 mar. 2014. Citado na página 14.

RUBNER, Y.; TOMASI, C.; GUIBAS, L. J. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, v. 40, n. 2, p. 99–121, 2000. Citado 2 vezes nas páginas 17 e 35.

SCHAFFER, C. Selecting a classification method by cross-validation. *Machine Learning*, v. 13, p. 135, 1993. Citado 2 vezes nas páginas 16 e 27.

THEODORIDIS, S.; KOUTROUMBAS, K. *Pattern Recognition*. 2. ed. United States of America: Elsevier, 2003. Citado 3 vezes nas páginas 15, 22 e 23.

TORGO, L. *Linguagem R - Programação para a Análise de Dados*. 1. ed. -: Escolar, 2009. Citado na página 19.

WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques*. 2. ed. Boston: Diane Cerra, 2005. Citado na página 19.