Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um estudo sobre agrupamento de sismos com técnicas de declustering

Gabriel Ferreira Silva

Monografia apresentada como requisito parcial
para conclusão do Projeto de Estudos em Inteligencia Artificial

Orientador
Prof. Dr. Marcelo Ladeira

Coorientador
Prof. Dr. Claus de Castro Aranha

Brasília
2015

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um estudo sobre agrupamento de sismos com técnicas de declustering

Gabriel Ferreira Silva

Monografia apresentada como requisito parcial
para conclusão do Projeto de Estudos em Inteligencia Artificial

Prof. Dr. Marcelo Ladeira (Orientador)
CIC/UnB

Prof. Dr. Claus Aranha
Department of Computer Sciences/University of Tsukuba

Prof. Dr. Guilherme Novaes Ramos
CIC/UnB

Brasília, 15 de dezembro de 2015

Dedicatória

Dedico este trabalho à minha família e aos meus amigos. Com vocês a vida é mais bonita.

Agradecimentos

Agradeço a Deus, por ter me dado saúde e disposição.

Agradeço a UnB, por ter me dado a oportunidade de desenvolver um projeto científico com pessoas bem qualificadas.

Agradeço ao ótimo grupo de trabalho do Projeto de Iniciação Científica (PROIC), formado por mim, Ícaro, Yuri, Ladeira e Claus. Eu achei nossas reuniões interessantes e divertidas, e os conselhos de vocês bastante úteis.

Agradeço ao meu orientador Marcelo Ladeira e ao meu coorientador Claus Aranha, pela oportunidade e pela disposição em me orientar.

Resumo

Para países com grande número de terremotos, a área de modelagem de riscos sísmicos é de importância fundamental. O GAModel é um modelo desenvolvido nessa área baseado em ideias da computação evolutiva, que obteve desempenho competitivo com outros métodos de avaliação de riscos sísmicos. Para isso, o GAModel utiliza dados de catálogos de terremotos, mas sem separar os terremotos principais dos terremotos secundários. Caso ocorresse tal separação, poderia-se trabalhar apenas com terremotos principais, ignorando os terremotos secundários. Com isso, seria esperado um desempenho ainda melhor por parte do GAModel, já que este trabalharia com dados livre de redundâncias. O objetivo deste trabalho é detalhar um plano para se aferir o impacto da separação de terremotos em terremotos principais e terremotos secundários na performance do GAModel, mostrando o que já foi feito até agora (tanto em termos de fundamentos teóricos obtidos quanto em termos de algoritmos implementados) e as próximas etapas.

Palavras-chave: Algoritmos Genéticos, Modelo de Riscos Sísmicos, *declustering*, *clustering*

Abstract

For countries with a large number of earthquakes, the field of risk assesment models is of uncontestable importance. The GAModel is a model developed on this area, based on ideias of evolutionary computation, that has obtained competitive results against the benchmarks. The GAModel works by using information contained on earthquakes data catalogs, but without separating earthquakes into mainshocks and aftershocks. If that separation happened, it would be possible to work only with the mainshocks, excluding the aftershocks from the analysis. After doing that, the results obtained by the GAModel should be even better, since the information would be free of redundancies. The goal of this work is to propose a plan to measure the impact of this separation in the performance of the GAModel, showing what has been done so far (in terms of teorethical foundations and in terms of coded algorithms) and the steps expected to the next semester.

Keywords: Genetic Algorithms, Seismic Prediction Models, declustering, clustering

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

EC Computação Evolutiva.

GA Algoritmos Genéticos.

JMA Agência Meteorológica Japonesa.

k-NN k-Nearest Neighbors.

UnB Universidade de Brasília.

Capítulo 1

Introdução

Nesse capítulo é apresentado o problema de *declustering*, o objetivo traçado no segundo semestre de 2015 e um resumo do que será realizado no primeiro semestre de 2016.

1.1 Definição do Problema

O conhecimento acerca de terremotos é fundamental para minimizar os impactos causados por tais fenômenos. Tal conhecimento é obtido a partir da análise de dados registrados anteriormente, que ficam organizados em catálogos de terremotos.

A partir desse catálogo, pode-se separar os terremotos em *mainshocks* (terremotos independentes) e *triggered earthquakes* (terremotos que são, ao menos parcialmente, dependentes de terremotos principais). Tal processo é importante para as pesquisas na área de avaliação de riscos sísmicos e na área de modelos de previsão de terremotos, e recebe o nome de *declustering* [6].

1.2 Objetivos

Este projeto de pesquisa tem por objetivo analisar a aplicabilidade dos métodos tradicionais de *declustering* para o problema de modelagem de risco de sismos. Espera-se ainda que os resultados obtidos deem suporte a pesquisa sobre modelo de riscos de sismos com técnicas de algoritmos genéticos, através da análise de como a performance de um desses modelos, o GAModel, é afetada quando é aplicado *declustering* ao catálogo de terremotos do qual o GAModel extrai informações.

Para a primeira parte do projeto, correspondente ao segundo semestre de 2015, objetivou-se adquirir os fundamentos teóricos que seriam necessários para o decorrer do projeto e se habituar com os *frameworks* que seriam utilizados durante o projeto. Para a segunda

parte do projeto, que corresponderá ao primeiro semestre de 2016, é esperado uma análise de como as técnicas de *declustering* melhoram a performance do GAModel.

Dependendo dos resultados obtidos, a aplicabilidade de técnicas tradicionais de *clustering* combinadas com ideias da Computação Evolutiva (EC, do inglês *Evolutionary Computation*) para o problema de *declustering* de terremotos será analisada. Os resultados obtidos serão apresentados no Congresso de Iniciação Científica da UnB.

Capítulo 2

Fundamentação Teórica

Neste capítulo são abordados os fundamentos teóricos (algoritmos genéticos, *clustering* e *cross-validation*) necessários para o desenvolvimento da pesquisa no futuro, que foram adquiridos durante o primeiro semestre de 2015. Para um melhor entendimento de como o GAModel funciona, estudou-se algoritmos genéticos. Para o problema específico de *declustering*, estudou-se o básico da área de *clustering*. Finalmente, para entender melhor como a validação de um modelo ocorre, estudou-se *cross-validation*.

2.1 Algoritmos Genéticos

O conteúdo dessa seção se encontram com mais detalhes no livro do Goldberg [4]. Os Algoritmos Genéticos (GA, do inglês *Genetic Algorithms*) são um campo de estudos da área de Inteligência Artificial, com muitas aplicações para problemas de busca em um espaço de soluções. Nessa abordagem, parte-se de um conjunto de soluções para um determinado problema e, com base na aplicação de operadores inspirados em ideias proveniente da biologia, uma melhor solução para o problema em questão é encontrada. Assim, pode-se considerar os algoritmos genéticos um método de busca heurística, que procuram no conjunto de soluções uma melhor solução.

2.1.1 Analogias com a biologia

Na área de algoritmos genéticos faz-se referência a muitos conceitos da biologia. Assim, para uma melhor compreensão sobre o assunto, deve-se explicar tais referências, e é o que fazemos a seguir. Uma solução para o problema em questão é chamada de indivíduo. Por consequência, um conjunto de soluções para um determinado problema é visto como um conjunto de indivíduos, recebendo então o nome de população. A qualidade de uma solução para a aplicação traduz-se como o quão bem adaptado o indivíduo está, e o

objetivo de encontrar a melhor solução tem por analogia encontrar o melhor indivíduo, i.e., o indivíduo mais adaptado, ou seja, aquele com maior *fitness*. Os algoritmos genéticos são baseados na ideia biológica de que, assim como indivíduos podem evoluir e se tornar cada vez mais bem adaptados, pode-se aplicar operadores a um conjunto inicial de soluções para um problema (ou seja, uma população) de modo a obter soluções cada vez melhores (ou seja, indivíduos cada vez mais adaptados). Tais operadores são descritos a seguir.

2.1.2 Principais operadores

Os principais operadores para a área de GA são *crossover*, seleção e mutação. Entretanto, antes de explicá-los, é necessário definir função de *fitness*. Conforme já dito, encontrar a melhor solução para o problema em estudo corresponde a encontrar o indivíduo mais bem adaptado. A função de *fitness* avalia justamente isso, associando a um indivíduo um valor numérico, que traduz o quão bem o indivíduo representa uma solução para a aplicação.

O primeiro operador a ser explicado é o de *crossover*. Tal operador gera um novo indivíduo (ou, dependendo do problema, dois novos indivíduos) a partir de dois indivíduos já existentes. Assim como na natureza o indivíduo gerado pelo cruzamento de dois pais tem em seu DNA uma combinação do DNA de seus genitores, em GA o indivíduo gerado pelo operador de *crossover* tem algumas características de cada um dos dois indivíduos que o produziram.

É importante perceber que ao gerarmos e avaliarmos um novo indivíduo estamos avaliando uma nova solução no espaço de busca, e que o indivíduo gerado por *crossover* pode ter *fitness* maior que seus pais ou não. Na representação mais simples e comum de GA, indivíduos são representados como *strings*. O crossover pode ser realizado então combinando partes das duas strings, conforme mostra a Figura 2.1.

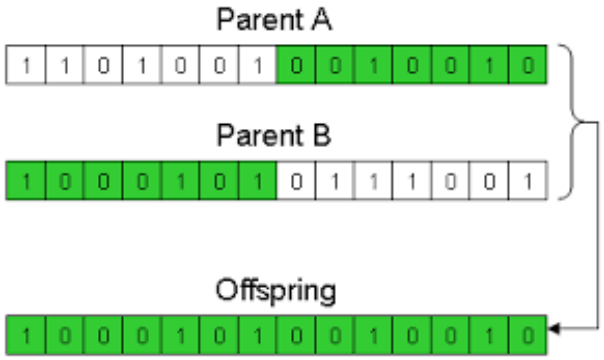

Figura 2.1: Crossover feito com a recombinação de partes das duas strings.

O segundo operador é o de seleção. Assim como na natureza os indivíduos mais bem adaptados tem maior probabilidade de gerar descendentes (Lei de Darwin), os algoritmos

genéticos devem valorizar e combinar as melhores soluções já encontradas, enquanto removem ou dão poucas chances às piores soluções, para que possam obter soluções cada vez melhores. O operador responsável por isso é o operador de seleção. Tal operador deve selecionar os indivíduos de acordo com a sua *fitness*, i.e., ao quão bem eles resolvem o problema em questão.

Por fim, o operador de mutação deve fazer pequenas modificações em um indivíduo, na esperança de que essas pequenas alterações gerem um indivíduo com maior *fitness*. A mutação é um operador importante, já que aumenta a variabilidade da população e contribui para evitar soluções que são simplesmente ótimos locais.

2.1.3 Funcionamento de um GA

Definidos os três operadores principais de GA, deve-se explicar como algoritmos genéticos funcionam de modo geral. Em geral, inicializa-se uma população com aproximadamente 100 indivíduos (tal número pode variar de acordo com a aplicação e a modelagem), escolhidos de modo aleatório no espaço de buscas.

Cada indivíduo tem sua *fitness* avaliada, e o melhor indivíduo é guardado em uma lista. Em seguida, aplica-se o operador de seleção, para selecionar os individuos que irao gerar descendentes. Depois, o operador de *crossover* é aplicado, gerando-se os descendentes da população. O operador de mutação pode então ser aplicado, aumentando a diferença (variabilidade) dos descendentes gerados em relação aos seus pais. Por fim, os descendentes gerados substituem (total ou parcialmente) a população que os gerou. Há várias técnicas para realizar essa substituição geracional: pode-se substituir apenas os N piores indivíduos da população, onde N é um parâmetro a ser especificado, pode-se garantir que o melhor indivíduo da população antiga não será nunca substituído (tal técnica se chama elitismo) ou pode-se substituir a população como um todo.

Tal processo é repetido várias vezes, para que várias soluções sejam exploradas. O número de vezes que o processo de gerar descentes a partir de uma população é repetido é conhecido como número de gerações, e fica tipicamente na casa das centenas. O melhor indivíduo gerado por todas as gerações é retornado, e corresponde a melhor solução encontrado pelo algoritmo genético.

Os algoritmos genéticos podem então ser vistos como um método de busca heurística. O fato de serem algoritmos estocásticos permite que busquem soluções variadas no espaço de busca a cada geração, enquanto que a heurística de combinar e valorizar as melhores soluções enquanto dão poucas chances a soluções piores torna-os computacionalmente eficientes para grandes volumes de dados. De fato, diversos estudos comprovam a riqueza de aplicações de GA.

2.2 *Clustering* e *Declustering*

2.2.1 *Clustering*

Clustering é a tarefa de agrupar uma coleção de dados, de modo que dados pertencentes a um mesmo grupo sejam o mais similar possível (tal similaridade depende da aplicação em questão), e sejam diferentes dos dados pertencentes a outro grupo. A Figura 2.2 ilustra a classificação de dados em 3 *clusters* diferentes.

Trata-se de uma campo de estudos com aplicações para as áreas de mineração de dados, aprendizagem de máquina, reconhecimento de padrões entre outras. Como há vários modelos para clustering, não há um melhor algoritmo para a tarefa de *clustering* , sendo a escolha do algoritmo dependente do contexto.

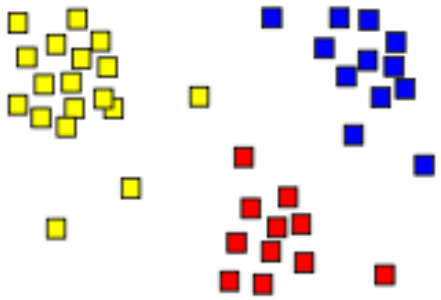

Figura 2.2: Agrupamento de dados em clusters.

2.2.2 *Declustering*

O conceito de *declustering* é detalhado no artigo do van Stiphout [6]. *Declustering*, quando analisado no contexto de previsão de sismos, corresponde a técnica de, a partir de um cátalogo de sismos, separar os terremotos em duas classes: *mainshocks* e *aftershocks*. Os *mainshocks* são terremotos independentes, enquanto que os *aftershocks*(também chamados de *foreshocks*, *offspring* ou *triggered earthquakes*) são terremotos parcialmente dependentes de algum *mainshock*. Assim, o nome *declustering* vem da possibilidade de, após separarmos os *mainshocks* dos *aftershocks*, removermos estes e usarmos apenas aqueles em um determinado problema, evitando assim informações redundantes. Este tratamento de dados é importante, tendo aplicações para as áreas de avaliação de riscos sísmicos e modelos de previsão de sismos [6].

Sabe-se que, para grandes regiões com ocorrências sísmicas, os *mainshocks* seguem uma distribuição de Poisson. Vários algoritmos, entretanto, conseguem separar *mainshocks* de

aftershocks, garantindo que os *mainshocks* sigam uma distribuição de Poisson. Assim, essa exigência apenas não é suficiente para avaliar o quão bom um método de *declustering* é, e diferentes métodos costumam ter performance diferente dependendo do domínio de aplicação [6]. Maiores detalhes sobre práticas de *declustering* podem ser encontradas no artigo do van Stiphout[6].

2.3 *Cross-validation*

2.3.1 Motivação para usar *Cross-validation*

Para o desenvolvimento de um modelo de previsão, geralmente dispomos de dados sobre condições passadas e seus resultados e desejamos que tal modelo seja capaz de fazer previsões acertadas sobre eventos futuros, que ocorrerão sob condições diferentes das que ocorreram no passado. Isso significa que queremos construir modelos que, através dos dados que dispomos, capturem a essência do fenômeno em questão para que consigam prevê-lo corretamente para dados ou condições que ainda não foram vistoss. Posto de outra forma, bons modelos de previsão devem ser capazes de generalizar bem para dados que não se encontram na amostra.

Um modelo de previsão que funcione muito bem para os dados da amostra mas que não generalize bem fora dela provavelmente não conseguiu separar o conteúdo contido nos dados do barulho que os dados podem eventualmente conter. Esse grande problema recebe o nome de *overfitting*. Para garantir que um determinado modelo não vá sofrer com *overfitting*, é comum separarmos os dados em dados de treino e dados de teste. Usamos os dados de treino para que o nosso sistema de previsão aprenda o fenômeno estudado e os dados de teste (os quais o nosso sistema não foi apresentado ainda) para ver se o modelo consegue generalizar bem, i.e, para ver se o modelo realmente aprendeu ou se apenas memorizou. A performance do sistema de previsão nos dados de teste fica sendo então a métrica final para avaliarmos seu desempenho [1].

Caso apenas uma divisão entre treino e teste fosse feita, a performance do nosso modelo dependeria das particularidades de como tal divisão particionou os dados entre treino e teste. Divisões sistemáticas em treino e teste podem evitar tal problema. Para conseguir isso, utilizam-se técnicas de *cross-validation*.

2.3.2 Técnicas comuns de *cross-validation*

As técnicas mais simples de *cross-validation* são o *K-fold* e o *Leave One Out*. No *K-fold*, a amostra é dividida em k grupos. k-1 grupos são usados como conjunto de treino e o último como conjunto de teste. Para cada grupo usado como teste o modelo tem

seu desempenho avaliado. No fim, uma avaliação do modelo pode ser feita ponderando sua performance para cada uma das k divisões possíveis. O *Leave One Out* funciona de maneira semelhante: os n pontos que compoem a amostra são divididos em treino e teste, com a restrição de que o teste é feito em apenas um ponto. Assim, tal técnica pode ser vista como um *K-fold* para o valor particular de k = n. Para a maioria das aplicações, trata-se de uma escolha ruim para o parâmetro k, devido ao alto custo computacional de se fazer n simulações.

Capítulo 3

Algoritmos Implementados e Resultados Obtidos

Conforme citado no capítulo 2, os fundamentos teóricos necessários como embasamento para a pesquisa constituem algoritmos genéticos, declustering e cross-validation. Durante o semestre alguns algoritmos relacionados a tais temas foram implementados, com o objetivo de melhorar o aprendizado destes conceitos. Para estudar *clustering* foi implementado o algoritmo do *k-means*. Para o estudo de GA, um algoritmo que utilizava conceitos desta área para fazer o agrupamento de dados foi implementado e seu desempenho foi comparado ao *k-means*. Por fim, para o estudo de *cross-validation*, estudou-se como duas separações diferentes em treino e teste alteram o desempenho medido do algoritmo classificador k-Nearest Neighbors (k-NN).

3.1 *k-means*

3.1.1 Definição do problema

Para entender mais como os algoritmos de *clustering* funcionam foi implementado um algoritmo simples de *clustering*: o *k-means algorithm*. O problema tem como entrada n observações e um número k de agrupamentos que se deseja formar, e tem como objetivo enquadrar tais observações em k *clusters*, onde k é um número conhecido pelo algoritmo e dependente do domínio em questão. A métrica mais comum para se determinar o quão bom é um conjunto de k *clusters* é obtida após se calcular, para cada ponto, sua distância (alternativamente, pode-se usar o quadrado da distância, atribuindo assim mais peso a *outliers*) em relação ao centróide do cluster a que ele pertence. Quanto menor a soma das distâncias calculadas melhor o agrupamento. Achar a solução correta para tal problema

é NP-Difícil, mas bons métodos heurísticos existem, como o algoritmo de Lloyd [5], que por ser tão comum as vezes é denominado *k-means algorithm*.

3.1.2 Algoritmo de Lloyd

O algoritmo de Lloyd é composto por duas etapas[5]: a de inicialização e a de refinamento. Na etapa de inicialização, k pontos são escolhidos para serem os centróides dos *clusters*. A maneira mais comum de se escolher esses k pontos é aleatoriamente, embora outros métodos existam. Decorrida essa etapa, entra-se na etapa de refinamento, composta por dois passos. O primeiro passo é atribuir cada ponto na amostra ao *cluster* mais próximo. O passo seguinte é calcular para cada *cluster* um novo centróide, já que novos pontos podem ter sido inseridos ou retirados dos *clusters*. Em seguida, volta-se para o primeiro passo e o processo é repetido várias vezes, terminando quando o passo de atribuir cada ponto ao *cluster* mais próximo não faz nenhum ponto mudar de *cluster*.

3.1.3 Resultados obtidos

Para testar se a implementação do *k-means* tinha sido feita de modo correto, foram feitas 3 simulações. Em cada simulação, pontos foram gerados de acordo com uma distribuição uniforme e classificados em *clusters*. A análise do resultado obtido consistiu de comparar cada um dos centróides obtidos para o conjunto de clusters fornecido pelo *k-means* com o centróide esperado teoricamente caso um número infinitamente grande de pontos fosse gerado. Um resumo dos resultados pode ser visto na Tabela 3.1. A seguir, detalhamos os parâmetros utilizados para cada simulação, os quais foram escolhidos de modo arbitrário.

Na primeira simulação, geraram-se pontos em duas dimensões. 10 desses pontos tinham suas coordenadas em x e em y variando uniformemente entre os valores -10 e 10. Outros 10 pontos tinham suas coordenadas em x e em y variando uniformemente entre 30 e 40. O algoritmo deveria então ser capaz de separar tais pontos em 2 *clusters*. Pode-se perceber então que, da maneira como os *clusters* foram escolhidos, um dos centróides obtidos deveria ter coordenadas próximas de (0, 0) e o outro de (35, 35). Executando a primeira simulação para o algoritmo implementado, obteve-se um centróide com coordenadas (2.11, 2.11) e outro com coordenadas (32.64, 30.90). Tais resultados mostram que o *k-means* conseguiu separar bem as observações em 2 *clusters*.

Na segunda simulação geraram-se pontos com 3 dimensões. 10 desses pontos tinham suas coordenadas em x, y e z variando uniformemente entre os valores -10 e 10. Outros 10 pontos foram gerados com coordenadas em x, y e z entre 30 e 40. O algoritmo deveria ser então capaz de separar tais pontos em 2 *clusters*. Pode-se perceber então que da maneira que como os *clusters* foram escolhidos, um dos centróides obtidos deveria ter coordenadas

Tabela 3.1: Resultado para as 3 simulações feitas para o k-means.

Número da Simulação	Centróide esperado	Centróide obtido
1	(0, 0)	(35, 35)
1	(2.11, 2.11)	(32.64, 30.90)
2	(0, 0, 0)	(-2.88, 0.33, 0.77)
2	(35, 35, 35)	(30.72, 31.09, 31.36)
3	(49.5, 49.5, 49.5, 49.5)	(15.33, 58.66, 24.66, 32.00)
3	(350, 350, 350, 350)	(341.69, 349.90, 345.90, 347.98)
3	(15, 15, 15, 15)	(29.85, 11.42, 39.57, 9.85)

próximas de (0, 0, 0) e o outro de (35, 35, 35). Executando tal simulação, obteve-se um centróide com coordenadas (30.72, 31.09, 31.36) e o outro com coordenadas (-2.88, 0.33, 0.77). Assim como na primeira simulação, os resultados acima mostram que o *k-means* conseguiu separar bem as observações em 2 *clusters*.

Na terceira simulação geraram-se pontos com 4 dimensões. 10 desses pontos tinham suas coordenadas em x, y, z e t variando uniformemente entre os valores -1 e 100. 100 pontos foram gerados com todas as suas coordenadas variando uniformemente entre os valores de 300 e 400. Além disso, 1 ponto foi gerado com as coordenadas x, y, z e t variando uniformemente entre os valores 10 e 20. O algoritmo deveria ser então capaz de separar tais pontos em 3 *clusters*. Pode-se perceber então que, da maneira como os *clusters* foram escolhidos, um dos centróides obtidos deveria ter coordenadas próximas de (49.5, 49.5, 49.5, 49.5), outro deveria ter coordenadas próximas de (350, 350, 350, 350) e o último deveria ter coordenadas próximas de (15, 15, 15, 15). Executando tal simulação, obteve-se um centróide com coordenadas (341.69, 349.90, 345.90, 347.98), outro com coordenadas (29.85, 11.42, 39.57, 9.85) e outro com coordenadas (15.33, 58.66, 24.66, 32.0). Pode-se perceber pelos resultados que o k-means obteve uma separação razoável.

Todas as simulações podem ser replicadas, já que o código do programa se encontra disponível na Internet [1]. Os resultados encontrados deverão ser parecidos, mas não idênticos, já que os pontos gerados variam de uma execução do programa para outra.

3.2 *Clustering* com GA

3.2.1 Modelagem e metodologia utilizada

Tentou-se usar GA para resolver o mesmo problema do algoritmo de Lloyd. A modelagem feita para o problema foi bem diferente do comum para algoritmos genéticos, sendo descrita a seguir. Cada indivíduo consistiu de um conjunto de k *clusters*, de modo que cada

[1] `https://github.com/gabriel951/scikit`

observação pertencesse a exatamente um *cluster*. A função de *fitness* foi então definida como sendo a soma do quadrado das distâncias de cada ponto ao centróide do cluster ao qual o ponto pertence. Para selecionar os indivíduos para o *crossover*, utilizou-se como método o *roullete whell*, em que a probabilidade de um indivíduo ser selecionado para *crossover* é proporcional a sua *fitness*. O *crossover* de dois indivíduos consistiu em parear cada *cluster* do primeiro indivíduo com o *cluster*do segundo indivíduo que estivesse mais próximo daquele. Para cada par, gerou-se um novo *cluster*, cujas coordenadas eram a média aritmética das coordenadas dos dois *clusters* que o geraram. Então, no fim, foram gerados k novos *clusters*, ou seja, um indivíduo. Por fim, o operador de mutação consistiu em pegar um ponto de um determinado *cluster* e colocá-lo no *cluster* vizinho a ele que estivesse mais próximo dele. Por fim, escolheram-se valores convencionais como parâmetros: 100 gerações, cada uma contendo 25 indivíduos e probabilidade de mutação sendo igual a 0.1. O *crossover* sempre foi feito. Por fim, deve-se descrever a metodologia utilizada. Para se testar o desempenho da modelagem utilizando algoritmos genéticos e compará-la ao algoritmo de Lloyd, foram feitas 3 simulações. Em cada simulação variou-se a quantidade de pontos utilizados, a dimensão dos pontos, o número de clusters e a distribuição que gerava os pontos. A dimensão dos pontos, o número de clusters e a distribuição que gerava os pontos foram escolhidos de modo arbitrário. Para cada simulação, cada um dos algoritmos recebeu um *score*, para avaliar o quão bem ele havia separado as observações. O *score* foi definido como sendo o inverso da soma do quadrado da distância de cada ponto ao *cluster* que ele pertence. Assim, quanto maior o *score* de um algoritmo menor é a soma das variâncias dentro de cada *cluster*. Os resultados obtidos são discutidos na seção seguinte.

3.2.2 Resultados obtidos

A Tabela 3.2 resume o desempenho do algoritmo genético frente o algoritmo de Lloyd, para cada uma das simulações feitas. O código dos dois algoritmos encontra-se na Internet[2], de modo que as simulações podem ser refeitas. Resultados próximos devem ser encontrados, mas não iguais, já que os pontos gerados serão diferentes e o algoritmo genético faz escolhas de modo estocástico.

Na primeira simulação, foram gerados 6 pontos com 2 dimensões. Os pontos foram agrupados em 3 *clusters*. O algoritmo genético e o algoritmo de Lloyd obtiveram o mesmo desempenho, como se pode observar pela Tabela 3.2. A performance igual dos algoritmos era esperada, pois o número de pontos em tal simulação foi bem pequeno.

Na segunda simulação, foram gerados 30 pontos com 4 dimensões. Os pontos foram agrupados em 3 *clusters*. Os resultados obtidos são mostrados na Tabela 3.2. A perfor-

[2] `https://github.com/gabriel951/scikit/blob/master/GA-means.py`

Tabela 3.2: Comparativo de performance entre algoritmo genético e algoritmo de Lloyd.

Número do teste	***Score* do algoritmo genético**	***Score* do algoritmo de Lloyd**
1	0.00359	0.00359
2	0.000248	0.000240
3	0.00002442	0.00002443

mance semelhante dos algoritmos era esperada, já que o número de pontos era pequeno. O fato do algoritmo genético ter *score* maior que o algoritmo de Lloyd mostrou que a abordagem de algoritmos genéticos e a modelagem utilizada podem ser competivas.

A terceira simulação objetivou verificar como os dois algoritmos reagiam quando o número de pontos era ligeiramente grande. Para isso, foram gerados 300 pontos com 4 dimensões. Os pontos foram agrupados em 3 *clusters*. Os resultados obtidos são vistos na Tabela 3.2. Pode-se notar que novamente o algoritmo genético teve desempenho competitivo com o algoritmo de Lloyd. Outro resultado interessante diz respeito ao tempo de execução de cada algoritmo: o algoritmo de Lloyd foi executado em menos de um minuto, enquanto que o algoritmo genético levou aproximadamente 10 minutos para executar. Esse tempo de execução sugere que a modelagem usada talvez não seja promisssora para um conjunto muito grande de pontos.

3.3 k-NN com *cross-validation*

3.3.1 Definição do problema e metodologia utilizada

Para entender o conceito de *cross-validation* e aplicá-lo em um contexto real, utilizou-se uma base de dados para reconhecimento de dígitos escritos à mão. As entradas dessa base de dados correspondiam a posição (tanto no eixo x quanto no eixo y) da caneta em 8 instantes de tempo distintos, e a saída correspondia ao dígito desenhado. Procurou-se desenvolver um sistema classificador, que, a partir de dados de treino aprendesse a classificar dígitos, e fosse capaz de classificar outros dígitos escritos a mão, correspondente ao conjunto de teste. No total, a base de dados é composta por 10 992 dígitos. Em seguida, fez-se uma redivisão dos dados de treino e de teste, e reavaliou-se a performance do sistema classificador. Com essa redivisão, pode-se avaliar como a divisão em treino e teste impacta a métrica da performace do algoritmo.

3.3.2 k-NN e sua aplicação no problema

O algoritmo *k-Nearest Neighbor*(k-NN) pode ser usado tanto para regressão quanto para classificação, tratando-se de um algoritmo simples de aprendizagem de máquina [2]. O

Tabela 3.3: Performance classificadora do k-NN para a divisão em treino e teste da scikit.

Valor de k	**Porcentagem de acerto na classificacao**
1	97.741%
2	97.369%
3	97.798%
4	97.655%
5	97.598%
6	97.570%
7	97.414%
8	97.414%
9	97.455%
10	97.512%
11	97.341%

k é um parâmetro passado para o algoritmo, e dependente do domínio em questão. No caso da classificação, que é o nosso foco, o algoritmo recebe como entrada, do conjunto de treino, os k exemplos de treino mais próximos e fornece como saída a classe do dado de teste que se deseja classificar. No nosso caso, um exemplo de treino consistiria de um vetor contendo a posição nos eixos x e y e a classe do dado de teste corresponde a qual dígito foi escrito.

3.3.3 Resultados obtidos

A base de dados em questão já havia separado dados de treino e dados de teste. Os dados de treino correspondiam a dígitos escritos por 30 voluntários e os dados de teste correspondiam a dígitos escritos por 14 voluntários diferentes. Avaliou-se então o desempenho do algoritmo k-NN da *scikit*[3], um *framework* da linguagem Python que é próprio para inteligência artificial, para o problema de classificar dígitos. Tal avaliação foi feita para diferentes escolhas do parâmetro k. Os resultados são mostrados na tabela Tabela 3.3.

Feita tal simulação, procurou-se investigar qual o impacto da separação em dados de treino e dados de teste. Para isso, fez-se nova simulação, com diferentes conjuntos de treino e teste. Para gerar os novos conjuntos de treino e teste, procedeu-se da seguinte maneira: juntou-se todos os dados (na base de dados da scikit eles já estavam separados em treino e teste) para posteriormente dividi-los aleatoriamente, colocando 75% deles no conjunto de treino e 25% deles no conjunto de teste. Avaliou-se novamente o desempenho do algoritmo k-NN, para diferentes valores de k. Os resultados obtidos são mostrados na Tabela 3.4. Os resultados mostram como a separação em dados de treino e dados de teste pode afetar como avaliamos a performance de sistemas classificadores. O melhor

[3] `http://scikit-learn.org/stable/`

Tabela 3.4: Performance classificadora do k-NN para a segunda divisão em treino e teste.

Valor de k	**Porcentagem de acerto na classificação**
1	99.672%
2	99.417%
3	99.417%
4	99.272%
5	99.490%
6	99.417%
7	99.417%
8	99.344%
9	99.344%
10	99.308%
11	99.235%

desempenho na segunda simulação era esperado, já que, enquanto na primeira simulação todos os dados de teste foram gerados por voluntários diferentes dos voluntários dos dados de treino, na segunda simulação os dados de teste poderiam ter sido feitos por voluntários que também tenham gerado dados de treino. Por fim, ressalta-se que o código executado para tais simulações encontra-se disponivel na Internet[4].

[4] `https://github.com/gabriel951/cross-validation`

Capítulo 4

Conclusão

O presente capítulo conclui sobre o trabalho realizado no segundo semestre de 2015 e traça as próximas etapas para o primeiro semestre de 2016.

4.1 Conclusão

O trabalho feito no primeiro semestre de 2015 visou a obtenção de conhecimentos básicos para o desenvolvimento da pesquisa em 2016. Para isso, foram estudados os tópicos: algoritmos genéticos, *declustering* e *cross-validation*. O estudo abordou tanto a teoria, através da leitura de materiais, quanto a parte prática, através da implementação de algoritmos. Após tal trabalho, é esperado que haja suficiente embasamento para o desenvolvimento da pesquisa no futuro, a qual tem suas próximas etapas detalhadas a seguir.

4.2 Próximas etapas

O trabalho a ser realizado tem como um de seus objetivos ajudar a pesquisa do aluno de graduação Yuri Cossich Lavinas, relacionado a criação de um modelo (o *GAModel*) baseado em algoritmos genéticos para previsão de riscos sísmicos. Em seu trabalho, foram usados dados de terremotos disponibilizados pela Agência Meteorológica Japonesa (JMA) para prever a localização de terremotos no Japão em um determinado ano. O modelo desenvolvido por ele teve desempenho competitivo com os *benchmarks* atuais, principalmente para áreas com grandes quantidades de atividades sísmicas que ocorreram no solo[3]. Entretanto, não houve uma separação prévia dos dados, de modo a separar *mainshocks* de *aftershocks* e eliminar estes, permitindo assim um trabalho livre de redundâncias. Avaliar se tal separação é possível e suas consequências são as próximas etapas do trabalho.

Para isso, serão aplicadas técnicas de *declustering* tradicionais aos catálogos de terremotos utilizados no projeto do Yuri. Medindo-se a performance do *GAModel* para os catálogos originais e para os catálogos após a remoção dos *aftershocks*, será possível ter uma ideia do quão significativo o processo de *declustering* é. A expectativa é que o GAModel funcione melhor caso se aplique declustering.

Dependendo dos resultados obtidos, futuras etapas podem incluir a busca da melhor combinação de parâmetros para um determinado algoritmo de declustering (possivelmente usando GA), o desenvolvimento de um modelo que use técnicas de *clustering* juntamente com ideias provenientes da Computação Evolutiva para se realizar o *declustering* e a análise de como o uso de técnicas de *declustering* afetam outros modelos de previsão de riscos sísmicos.

Referências

[1] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, e Hsuan-Tien Lin. *Learning from Data*. 2012. 7

[2] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 1992. 13

[3] Aranha C., Lavinas Y., Ladeira M., e Enescu B. Is it possible to use generate good earthquake risk models using genetic algorithms. *Proceedings of the International Conference on Evolutionary Computation Theory and Applications (ECTA-2014)*, 2014. 16

[4] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. 3

[5] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, e Angela Y Wu. An efficient k-means clustering algorithm: Analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):881–892, 2002. 10

[6] van Stiphout, T., J. Zhuang, e D. Marsan. Seismicity declustering, 2012. 1, 6, 7