Trabalho Computacional 02
TIP8311 - Reconhecimento de Padrões
Quantização Vetorial na Detecção de Omissões de Pagamento

Artur Rodrigues Rocha Neto - 431951
Mestrando em Engenharia de Teleinformática
artur.rodrigues26@gmail.com

7 de Dezembro de 2018

1 Introdução

Dados de abril de 2017 do Banco Central do Brasil [1] indicam que, àquela época, quase 40% dos usuários de cartão de crédito ou não pagaram o valor mínimo ou atrasaram suas parcelas em mais de 90 dias. Um indivíduo que não cumpre com suas despesas financeiras passa a ter dificuldade em pedir empréstimos e outros benéficios junto a bancos, por exemplo. Entretanto, o aumento no índice de inadimplência gera gera incertezas no mercado, diminuindo a confiança de insituições financeiras e criando um desafio para os sistemas de aval de crédito.

Tendo em vista esse cenário, faz-se necessário modelos que ajudam a prever o pagamento ou não de faturas de cartão como uma medida preventiva à situação do mercado. Vários trabalhos exploram diferentes modelos preditivos, como regressão e classificação, para a analisar os hábitos de consumo e pagamento de indivíduos com aquele fim [2, 3].

Este trabalho usa um conjunto de dados[1] com dados econômicos e demográficos de usuários de cartão de crédito afim de montar tais modelos de predição. O objetivo é estudar esse conjunto de dados e aplicar técnicas e algoritmos que auxiliem na criação de vetores de característica satisfatórios para uso em classficadores.

2 Conjunto de Dados

O conjunto de dados `default-of-credit-card-clients.csv` agrega $N = 30000$ amostras com $p = 23$ atributos, onde cada amostra representa o perfil de compra de um cidadão de Taiwan. Os atributos perfazem informações demográficas (e.g. gênero, idade, escolaridade) e financeiras (e.g. histórico de pagamentos, valor de faturas passadas), estas coletadas durante um período de 6 meses. [2]. A coluna `default-payment-next-month` guarda zero (0) se aquele indivíduo pagou a fatura do cartão no mês seguinte ou um (1) caso ele não tenha pago. Uma visão estatística geral dos atributos vem listada na Tabela 1.

[1] `http://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients`

Atributo	Média	Mediana	Min	Max	Desvio
LIMIT_BAL	167484.32	140000.0	10000	1000000	129747.66
SEX	1.6	2.0	1	2	0.49
EDUCATION	1.85	2.0	0	6	0.79
MARRIAGE	1.55	2.0	0	3	0.52
AGE	35.49	34.0	21	79	9.22
PAY_0	-0.02	0.0	-2	8	1.12
PAY_2	-0.13	0.0	-2	8	1.2
PAY_3	-0.17	0.0	-2	8	1.2
PAY_4	-0.22	0.0	-2	8	1.17
PAY_5	-0.27	0.0	-2	8	1.13
PAY_6	-0.29	0.0	-2	8	1.15
BILL_AMT1	51223.33	22381.5	-165580	964511	73635.86
BILL_AMT2	49179.08	21200.0	-69777	983931	71173.77
BILL_AMT3	47013.15	20088.5	-157264	1664089	69349.39
BILL_AMT4	43262.95	19052.0	-170000	891586	64332.86
BILL_AMT5	40311.4	18104.5	-81334	927171	60797.16
BILL_AMT6	38871.76	17071.0	-339603	961664	59554.11
PAY_AMT1	5663.58	2100.0	0	873552	16563.28
PAY_AMT2	5921.16	2009.0	0	1684259	23040.87
PAY_AMT3	5225.68	1800.0	0	896040	17606.96
PAY_AMT4	4826.08	1500.0	0	621000	15666.16
PAY_AMT5	4799.39	1500.0	0	426529	15278.31
PAY_AMT6	5215.5	1500.0	0	528666	17777.47

Tabela 1: Estatísticas gerais dos atributos

A Figura 1 mostra a proporção entre as amostras de cada classe. Existe uma grande desequilíbrio entre o número de observações, sendo a quantidade de amostras indicando omissão de pagamentos igual a 23364 e aquelas pagas, 6636. Esse é um problema comum em muitos problemas de classificação que pode gerar consequências indesejáveis, como:

1. A definição de um conjunto de testes fica compremetida, pois a técnica de amostragem usada pode desfavorecer a classe desbalanceada.

2. Quando usado em tempo real, o modelo pode ficar não-otimizado na tarefa de classificação de novas amostras da classe desbalanceada.

Conjuntos de dados dessa natureza tendem a se valer basta da quantização, principalmente se poder reduzir o conjunto de amostras pagas para um valor semelhante ao de não pagas, tornando os conjuntos de treinamento e teste mais equilibrados.

A Figura 2 traz o perfil de distribuição dos 23 atributos do conjunto. É possível notar que, em sua maiorias, os preditores não respeitem distribuições normais. Alguns desses atributos são categóricos (sexo, escolaridade, estado cívil e idade) e devem ser removidos da análise ou trocados por *dummy features*. É provável que uma transformação de remoção de assimetria ajude no desempenho do Classificador Quadrático Gaussiano. Já a diferença de escala entre os valores é considerável, tornando indicada a aplicação de normalização.

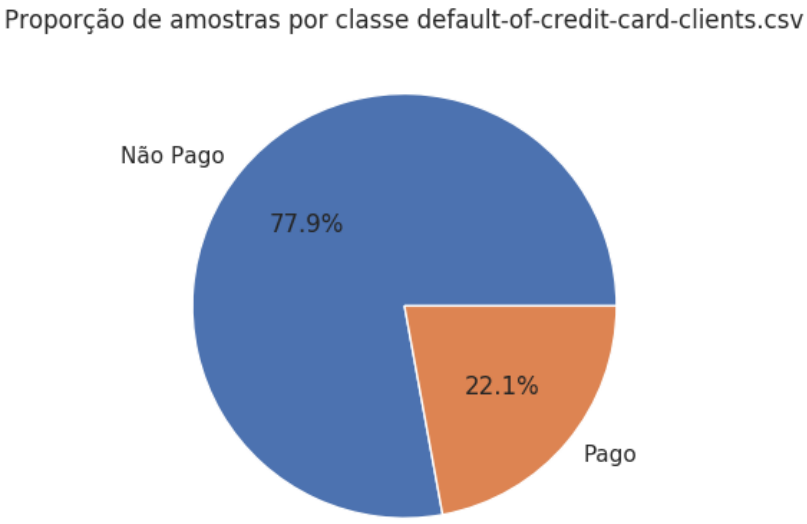

Figura 1: Número de amostras por classe

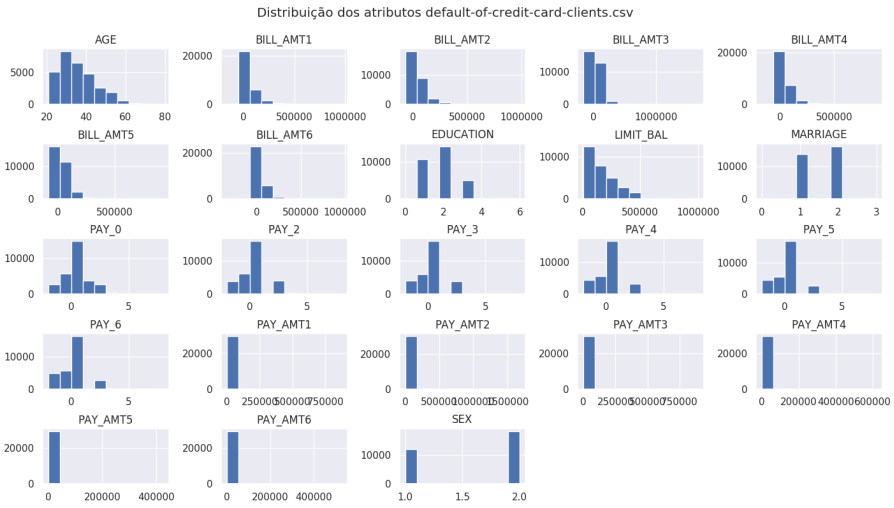

Figura 2: Distribuição dos atributos

O mapa de calor na Figura 3 (em valores absolutos) mostra a correlação entre os pares de atributos do conjunto. Algumas colinearidades estão presentes no subconjunto de atributos que diz respeito a atrasos no pagamento.

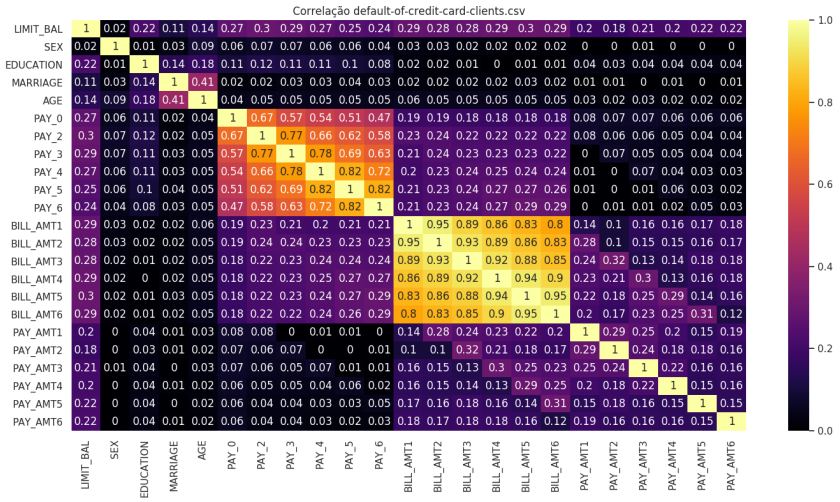

Figura 3: Correlação dos atributos

3 Metodologia

Foram comparados os desempenhos de três classificadores: Vizinho mais Próximo (NN), Distância Mínima ao Centróide (DMC) e Classificador Quadrático Gaussiano (CQG). Dois cenários de amostragem foram escolhidos: um com os dados originais e outro com uma redução de dimensionalidade usando K-médias. Cada classificador foi executado 100 vezes. A comparação de desempenho foi realizada em termos de média, mediana, valores mínimo/máximo, desvio padrão, sensibildiade e especificidade. O ambiente de experimentação foi implementado em Python 3.5 com o auxílio da biblioteca de aprendizagem de máquina ***scikit-learn*** [4]. Todos os testes foram executados em uma máquina Debian GNU/Linux 9.5 com 8GB de RAM e processador i7-3770 @3.40GHz x4.

3.1 Algoritmo de Clusterização k-médias

O K-médias é um método de clusterização (ou agrupamento) que arranja massas de dados em n conjuntos bem separados e de igual variância. Seu funcionamento baseia-se na minimização de um critério chamado *soma das distâncias quadráticas*, do inglês *Squared Sum Distance* ou apenas SSD. O índice SSD é conhecido também como inércia, o que facilita no seu entendimento: quanto menor a inércia de um ponto, menos esse ponto "se moveu"de uma interação a outra. A inércia, portanto, é o critério de convergência do K-médias. O algoritmo do K-médias pode ser descrito com os seguintes passos [5]:

Passo 1: Definir um valor para K.
Passo 2: Atribuir valores iniciais aos K protótipos.
Passo 3: Determinar a partição V_i do protótipo w_i, $i = 1, 2, ..., K$, unsando a Eq.1:

$$V_i = \{x \in \mathbb{R}^p | ||x - w_i|| < ||x - w_j||, \forall j \neq i\} \quad (1)$$

Passo 4: Calcular a nova posição do protótipo w_i como a média dos N_i objetos

da partição V_i:

$$w_i = \frac{1}{N_i} \sum_{x \in V_i} x \tag{2}$$

Passo 5: Repetir os Passos 3 e 4 até a convergência do algoritmo.

A implementação do K-médias no pacote ***scikit-learn*** encontra-se na classe `sklearn.cluster.KMeans`.

3.2 Classificador Vizinho Mais Próximo (NN)

O classificador Vizinho Mais Próximo (NN) é um classificador simples, porém muito poderoso. Baseado em distâncias, seu algoritmo consiste em [6]:

Passo 1: Armazenar em memória o conjunto de treinamento X

Passo 2: Para cada nova observação x, buscar em X pelo índice do vetor de atributos mais próximo de x:

$$i^* = arg \min_{i=1,...,N} ||x - x_i||^2 \tag{3}$$

onde $||v||$ denota a norma euclidiana do vetor v.

Passo 3: Atribuir à observação x o índice de x_{i^*}

A Eq.3 pode ser modificada para usar normas de qualquer ordem, não apenas a euclidiana. Uma outra norma muito usada é a Manhattam (Minkowski de ordem 1), mostrada na Eq.4:

$$d(x, x_i) = \sum_{i=1}^{N} |x - x_i| \tag{4}$$

A implementação do NN no ***scikit-learn*** traz diversas otimizações, incluindo o particionamento do espaço de atributos usando *KD-Tree* [7] ou *Ball-Tree* [8] para acelerar a busca do vizinho mais próximo. A implementação do NN está contida na classe `sklearn.neighbors.KNeighborsClassifier`.

3.3 Classificador Distância Mínima ao Centróide (DMC)

O classificador Distância Mínima ao Centróide tem um funcionamento muito semelhante ao NN. A ideia agora é calcular a distância de uma nova observão aos centróides das massas de dados de cada classe. Dada uma nova observação x a ser classificada, o algoritmo faz:

Passo 1: Encontrar o vetor centróide de cada classe:

$$m_i = \frac{1}{N_i} \sum_{\forall x \in \omega_i} x \tag{5}$$

onde N_i é o número de amostras da i-ésima classe, cujo rótulo é ω_i, para $i = 1, ..., K$

Passo 2: Rotular a nova observação x como da classe de m_{i^*} se:

$$||x - m_{i^*}||^2 < ||x - m_i||^2, \forall i \neq i^* \tag{6}$$

onde $||v||$ denota a norma euclidiana do vetor v.

Assim como no NN, outras normas podem ser usadas para o cálculo da distância no classificador DMC. A distância é uma métrica de dissimilaridade e o conhecimento de suas diversas formas e respectivas interpretações geométricas é chave para a construção de modelos preditivos robustos [6]. A implementação do DMC está contida na classe `sklearn.neighbors.NearestCentroid`.

3.4 Classificador Quadrático Gaussiano (CQG)

A Regra de Bayes, ponto de partida para a demonstração do CQG, responde à seguinte pergunta: "de posse dos exemplos já observados X, qual a probabilidade de uma nova observação x pertencer à classe w_i, ou seja, quanto vale $p(w_i|x)$?"[9].

Seja $p(w_i)$ a probabilidade *a priori* de uma observação pertencer a w_i, $p(x)$ a probabilidade dos atributos e assumindo que os exemplos respeitam uma distribuição de probabilidade normal multivariada $p(x|w_i)$ de média μ_i e matriz de covariância Σ_i (também chamada função de verossimilhança), temos [10]:

$$p(x|w_i) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma_i|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(x-\mu_i)^T\Sigma_i^-1(x-\mu_i)\right\} \tag{7}$$

$$p(w_i|x) = \frac{p(w_i)p(x|w_i)}{p(x)} \tag{8}$$

A classificação da nova observação x pode ser feita usando a regra da *máxima a posteriori*, ou critério MAP: x pertencerá à classe w_j se $p(w_j|x)$ for a maior densidade entre todas as K classes, ou seja:

$$w_j = arg\max_{i=1,...,K} p(w_i|x) \tag{9}$$

Uma forma mais geral do critério MAP pode ser escrita a partir de qualquer função que retorne o grau de pertinência da observação x a uma classe w_i, chamada *função discriminate* $g_i(x)$. Uma função discriminate muito utilizada é derivada a partir do logaritmo natural aplicado à Eq.7:

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T\Sigma_i^-1(x-\mu_i) - \frac{1}{2}\ln|\Sigma_i| + \ln p(w_i) \tag{10}$$

Assumindo que as classes são equiprováveis ($p(w_1) = p(w_2) = ... = p(w_k)$), o termo $\ln p(w_i)$ se torna irrelevante. Distribuindo os termos restantes do lado direito da Eq.10, temos:

$$\begin{aligned}
g_i(x) &= -\frac{1}{2}(x-\mu_i)^T\Sigma_i^-1(x-\mu_i) - \frac{1}{2}\ln|\Sigma_i| \\
g_i(x) &= -\frac{1}{2}[x^T\Sigma_i^-1x - x^T\Sigma_i^-1\mu_i - \mu_i^T\Sigma_i^-1x + m_i^T\Sigma_i^-1\mu_i] - \frac{1}{2}\ln|\Sigma_i| \\
g_i(x) &= -\frac{1}{2}[x^T\Sigma_i^-1x - 2\mu_i^T\Sigma_i^-1x + \mu_i^T\Sigma_i^-1\mu_i] - \frac{1}{2}\ln|\Sigma_i|
\end{aligned} \tag{11}$$

Podemos ver que a derivação final da Eq.11 representa a equação de um hiperparabolóide. Logo, o CQG não é um classificador linear, e sim quadrático como o próprio nome sugere. A invertibilidade da matriz Σ_i é um ponto crítico no uso desse classificador, logo recomenda-se o uso de alguns métodos para avaliar a sua invertibilidade e, em caso de necessidade, estimação da inversa. A classe `sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis` contém a implementação do CQG.

3.5 Normalização

De forma a melhorar a estabilidade de certos cálculos, um dos passos de pré-processamento mais simples e utilizado em *pipelines* de predição é a normalização, também conhecida como escalamento ou padronização [9].

Para normalizar um conjunto de dados, primeiro fazemos uma transformação de centralização, subtraindo de cada valor de um atributo k a média dos valores do mesmo $\bar{x_k}$. Depois, uma operação de escala é efetuada dividindo todos os valores de cada atributo x_k pelo desvio padrão σ_k. Ambos os passos podem ser descritos com uma fórmula, mostrada em Eq.12.

$$x_k^* = \frac{x_k - \bar{x_k}}{\sigma_k} \tag{12}$$

A transformação de normalização é implementada no ***scikit-learn*** pela classe `sklearn.preprocessing.StandardScaler`.

3.6 Remoção de Assimetria

Um distribuição de dados é dita simétrica quando a probabilidade de um evento ocorrer de um lado da distribuição é praticamente a mesmo que na região equidistante oposta. A assimetria de uma distribuição é cáculada usando a Eq.13, onde x é o atributo, n o número de valores e $\bar{x}$ é a média do atributo. Um distribuição possui *assimetria à direita* quando apresenta uma maior densidade de valores no lado esquerdo que do lado direito, e o seu valor é positivo. O análogo também serve para a *assimetria à esquerda* e o seu valor é negativo.

$$\begin{aligned} assimetria &= \frac{\sum (x_i - \bar{x})^3}{(n-1)v^{3/2}} \\ v &= \frac{\sum (x_i - \bar{x})^2}{(n-1)} \end{aligned} \tag{13}$$

Existe uma família de transformações que podem ser usadas para remover assimetria de dados. Aquela escolhida para esse trabalho foi a Yeo-Johnson [11], mostrada na Eq.14, onde λ é um parâmetro que define a transformação (pode ser estimado a partir de conjunto de treinamento [9]):

$$x^* = \begin{cases} ((x+1)^\lambda - 1)/\lambda & \text{se } \lambda \neq 0,\ x \geq 0 \\ \log(x+1)\text{se} & \text{se } \lambda = 0,\ x \geq 0 \\ -[(-x+1)^{2-\lambda} - 1]/(2-\lambda) & \text{se } \lambda \neq 2,\ x < 0 \\ -\log(-x+1) & \text{se } \lambda = 2,\ x < 0 \end{cases} \tag{14}$$

A transformação de remoção de assimetria é implementada no ***scikit-learn*** pela classe `sklearn.preprocessing.PowerTransformer`.

3.7 Avaliação de Performance dos Classificadores

O objetivo deste trabalho é o diagnóstico ou não da Doença de Parkinson. Portanto, podemos dizer que trata-se de um problema de classificação binária. Podemos agrupar o resultado de um conjunto de predições de um dado classificador na forma de uma Matriz de Confusão (Tabela 2). A matriz de confusão agrega de forma gráfica a quantidade de eventos preditos correta e incorretamente. Podemos extrair:

		Condição Real		
		Parkinson (1)	Saudável (0)	Total
Predição	Parkinson (1)	VP	FP	VP + FP
	Saudável (0)	FN	VN	FN + VN
	Total	VP + FN	FP + VN	N

Tabela 2: Matriz de confusão e seus elementos

- **Verdadeiros Positivos (VP):** pacientes com Parkinson corretamente diagnosticados com a doença;
- **Verdadeiros Negativos (VN):** pacientes saudáveis corretamente diagnosticados sem a doença;
- **Falsos Positivos (FP):** pacientes saudáveis diagnosticados incorretamente com a doença; e
- **Falsos Negativos (FN):** pacientes com Parkinson dignosticados incorretamente como saudáveis.

Precisão é a taxa de acerto geral (Eq.15). Sensibilidade mede o quanto as observações Verdadeira Positivas foram corretamente classificadas como tal (Eq.16). Especificidade é o análogo da sensibilidade, mas em relação aos Verdadeiros Negativos (Eq.17).

$$A = \frac{VP + VN}{VP + VN + FP + FN} \tag{15}$$

$$SENS = \frac{VP}{VP + FN} \tag{16}$$

$$SPEC = \frac{VN}{VN + FP} \tag{17}$$

A modelagem em classes do ***scikit-learn*** implementa, para todos os classificadores, um método `fit()` para treinamento, `score()` para teste e `predict()` para predição de novas observações. A função `sklearn.metrics.confusion_matrix` foi usada para extrair os valores de VP, VN, FP e FN.

4 Resultados

Apresentaremos agora os resultados comparativos entre os classificadores escolhidos, com e sem o uso do K-médias na redução de amostras. Dois *scripts* Python foram criados: `recpad.py`, que agrega diversas funções utilitárias para classificação e visualização, e `tc2.py`, sequência do passo-a-passo de geração de resultados deste trabalho. Pedaços principais de código serão apresentados ao longo dos resultados. Para uma avaliação das implementações, ver arquivo `recpad.py`.

Todas as experimentações foram repetidos 100 vezes, os resultados de cada rodada armazenados e um conjunto de estatísticas, analisado. A proporção 70/30 entre amostras de treinamento e teste foi mantida constante em todos os cenários abordados.

```
from recpad import *

classifiers = {"NN"   : NN(n_neighbors=1),
               "DMC"  : DMC(),
               "CQG"  : CQG(store_covariance=True)}
dataset = "data/default-of-credit-card-clients.csv"
to_drop = ["SEX", "EDUCATION", "MARRIAGE", "AGE"]
test_size = 0.3
rounds = 100
default_n_clusters = 1000

df = pd.read_csv(dataset)
df = df.drop(["ID"], axis=1)
df = df.drop(to_drop, axis=1)
X = df.drop(["default-payment-next-month"], axis=1)
y = df["default-payment-next-month"]

X_c0 = df.loc[df["default-payment-next-month"] == 0]
X_c0 = X_c0.drop(["default-payment-next-month"], axis=1)
X_c1 = df.loc[df["default-payment-next-month"] == 1]
X_c1 = X_c1.drop(["default-payment-next-month"], axis=1)

X_trans = super_normalize(X)
X_trans = super_unskew(X)
X_trans = pd.DataFrame(X_trans, columns=X.columns)

print("Amostras da classe 0: {}".format(X_c0.shape))
print("Amostras da classe 1: {}".format(X_c1.shape))
```

Listing 1: Cabeçalho do código trabalho02

4.1 Resultados Preliminares

Primeiro, exploramos o desempenho dos classificadores sem redução e com uma única configuração de agrupamento para $K = 1000$. Os resultados foram organizados na Tabela 3. As três últimas linhas, iniciadas com o nome do classificador e +km, mostram os valores para o K-médias.

Antes da redução, o NN apresentava a melhor taxa de acerto $(69,46\%)$, porém com um nível de sensibilidade bastante baixo $(29,47\%)$. O CQG, por sua vez, se mostrou a pior escolha de classificador; baixa taxa de acerto e maior desvio padrão dentre os três.

Após o uso do K-médias, o cenário se inverteu. O CQG foi o classificador que mais se valeu da redução, alcançando uma taxa matematicamente igual ao

Classif	Média	Mediana	Min/Max	Desvio	Sens	Espec
NN	69,46	69,41	68,42/70,72	0,44	29,47	80,82
DMC	53,63	53,56	52,64/54,63	0,44	67,08	49,81
CQG	49,89	49,18	40,5/69,54	5,92	82,33	40,66
NN+km	39,18	39,25	34,83/43,50	1,75	37,32	41,07
DMC+km	61,47	61,08	56,83/65,83	1,60	66,63	55,41
CQG+km	69,47	69,33	66,33/73,50	1,55	89,92	49,06

Tabela 3: Desempenho dos classificadores sem e com clusterização ($K = 1000$)

do NN sem redução: $69,47\%$. Já o NN perdeu desempenho, caiu para $39,18\%$ de precisão média e razão sensibilidade/especificidade abaixo dos $50,00\%$.

```
ans = classify(classifiers, X, y, test_size, rounds)
sumary(ans, "TC2 - classificacao sem reducao de dados")

print("classe 0...")
X_c0_red, inertia0 = reduction_kmeans(X_c0, n=1000)
print("classe 1...")
X_c1_red, inertia1 = reduction_kmeans(X_c1, n=1000)
X_red = np.concatenate((X_c0_red, X_c1_red))
y_c0 = np.zeros((1000, ), dtype=int)
y_c1 = np.ones((1000, ), dtype=int)
y_red = np.concatenate((y_c0, y_c1))
print("classificando...")
ans = classify(classifiers, X_red, y_red, test_size, rounds)
sumary(ans, "TC2 - desempenho com kmedias K=1000")
```

Listing 2: Resultados preliminares

4.2 Explorando valores de K

Feita uma primeira exploração, partimos para uma investigação mais aprofundada do algoritmo K-médias. Foi usada uma sequência de 30 valores de K protótipos, de 100 a 3000 com intervalos de 100, para criar uma gama maior de conjuntos de treinamento e teste. O resultado obtido encontra-se na Figura 4. Podemos tecer alguns pontos com base nesse resultado:

1. O CQG atinge um máximo de desempenho ($82,45\%$) para um agrupamento pequeno ($K = 100$)
2. A partir de $K = 500$, todos os classificadores apresentam uma evolução da precisão com o aumento do número de amostras
3. Após $K = 1000$, a evolução no desempenho do CQG e DMC é bastante sutil, enquanto que o NN melhora de maneira mais acentuada

A Figura 5 mostra o valor do SSD na última interação de cada executação do K-médias para cada K testado. Vemos que a inércia da classe 1 para K é pequenos consideravelmente menor que a da classe 0. Isso se deve ao fato da classe 1 possuir menos amostras, tornando mais fácil a coesão do agrupamento formado. Vemos que para valores de K altos, a inércia de ambas tende a se aproximar, pois a diferença no número de amostras passa a não ser tão relevante.

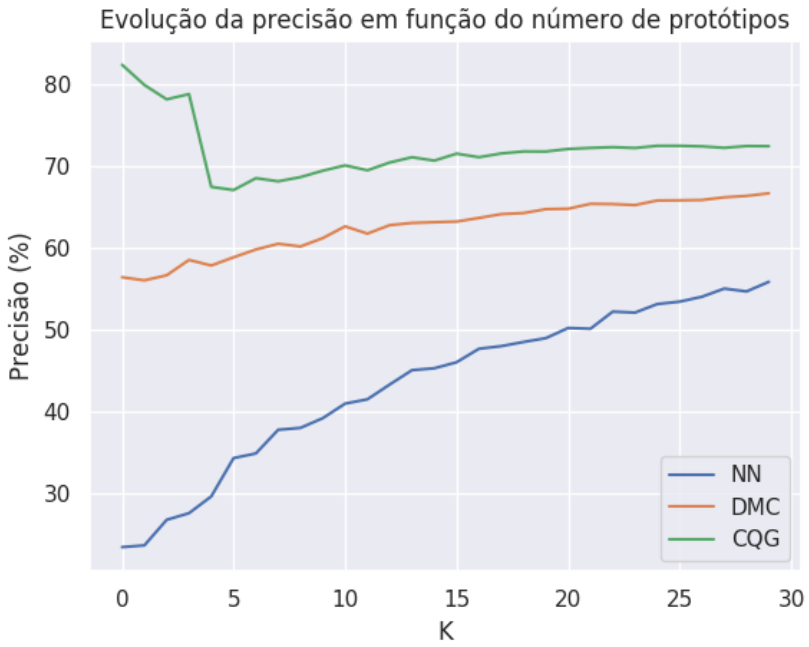

Figura 4: Taxa média de precisão dos classificadores em função de K

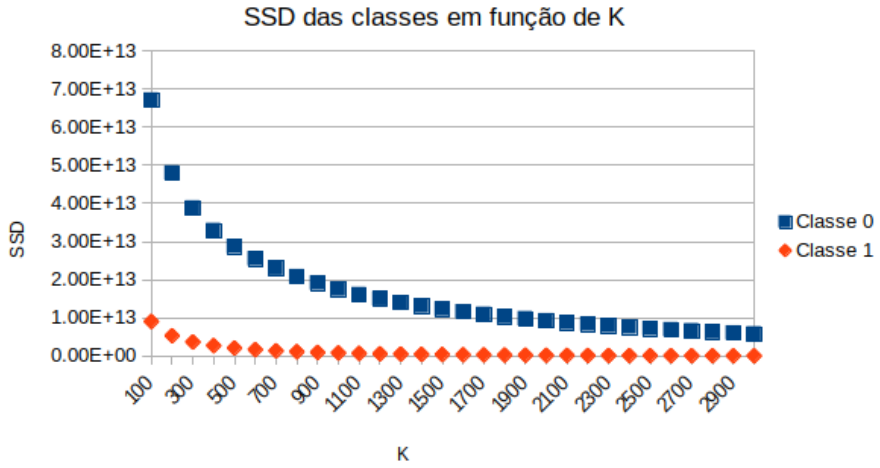

Figura 5: Índice SSD para cada classe em função de K

```
data = {"NN" : [], "DMC" : [], "CQG" : []}
inertias = {"inertia0" : [], "inertia1" : []}
n_init = 100
n_end = 3100
n_step = 100
ticks = int((n_end - n_init) / n_step)
vector = range(n_init, n_end, n_step)
```

```
8
9  print("Numero de amostragens: {}".format(ticks))
10 for n in vector:
11     print("kmedias para reducao (num_prototipos = {})".format(n))
12
13     print("classe 0...")
14     X_c0_red, inertia0 = reduction_kmeans(X_c0, n=n)
15
16     print("classe 1...")
17     X_c1_red, inertia1 = reduction_kmeans(X_c1, n=n)
18
19     X_red = np.concatenate((X_c0_red, X_c1_red))
20
21     y_c0 = np.zeros((n, ), dtype=int)
22     y_c1 = np.ones((n, ), dtype=int)
23     y_red = np.concatenate((y_c0, y_c1))
24
25     print("classificando...")
26     ans = classify(classifiers, X_red, y_red, test_size, rounds)
27
28     nn = round(np.mean(ans["NN"]["score"])*100, 2)
29     dmc = round(np.mean(ans["DMC"]["score"])*100, 2)
30     cqg = round(np.mean(ans["CQG"]["score"])*100, 2)
31
32     data["NN"].append(nn)
33     data["DMC"].append(dmc)
34     data["CQG"].append(cqg)
35     inertias["inertia0"].append(inertia0)
36     inertias["inertia1"].append(inertia1)
37
38     print("[n={} OK]:nn={},dmc={},cqg={}".format(n, nn, dmc, cqg))
```

Listing 3: Explorando valores de K

4.3 Desempenho com Dados Pré-Processados

Foram efetuadas transformações de normalização e no conjunto de dados originais e então repetidos os passos vistos na Seção 4.1. A Figura 6 mostra como ficaram os atributos pós pré-processamento. Os resultados gerais estão na Tabela 4.

Podemos constatar como a transformação de remoção de assimetria melhorou a performance do classificador quadrático CQG. Tecnicamente empato com o NN, aparece como o melhor classificador, obtendo taxa média de acerto de $73,29\%$. Para os dados transformados com $K = 100$, vemos uma queda nos resultados em comparação com a Tabela 3, com excessão do NN (precisão de $49,97\%$).

Classif	Média	Mediana	Min/Max	Desvio	Sens	Espec
NN	73,10	73,06	72,38/74,26	0,37	38,83	82,81
DMC	67,25	67,24	66,18/68,39	0,45	63,06	68,43
CQG	73,29	73,56	68,29/75,24	1,35	59,80	71,11
NN+km	49,97	50,17	46,00/53,50	1,70	43,25	56,74
DMC+km	52,07	52,08	48,67/56,00	1,46	45,31	58,84
CQG+km	52,60	52,50	47,33/63,83	2,76	86,00	19,14

Tabela 4: Desempenho dos classificadores nos dados pré-processados, sem e com clusterização ($K = 1000$)

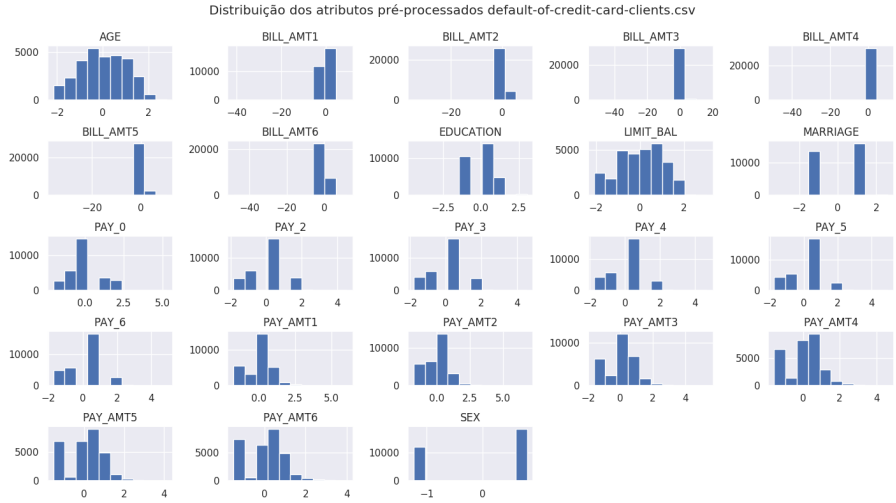

Figura 6: Distribuição dos dados pré-processados

```
X_c0_trans = super_normalize(X_c0)
X_c0_trans = super_unskew(X_c0_trans)
X_c1_trans = super_normalize(X_c1)
X_c1_trans = super_unskew(X_c1_trans)
print("classe 0...")
X_c0_red, inertia0 = reduction_kmeans(X_c0_trans, n=1000)
print("classe 1...")
X_c1_red, inertia1 = reduction_kmeans(X_c1_trans, n=1000)
X_red = np.concatenate((X_c0_red, X_c1_red))
y_c0 = np.zeros((1000, ), dtype=int)
y_c1 = np.ones((1000, ), dtype=int)
y_red = np.concatenate((y_c0, y_c1))
print("classificando...")
ans = classify(classifiers, X_red, y_red, test_size, rounds)
sumary(ans, "TC2 - desempenho com kmedias K=1000 e
    pre-processamento")
```

Listing 4: Redução em conjunto pré-processado

5 Conclusões

A redução dos dados usando K-médias revelou o potencial de classificação do CQG. Sua evolução em comparação com o cenário de dados completos foi a mais expressivas dentre todos os classificadores testados. O uso do K-médias original pode se revelar custoso e, mesmo com computadores relativamente potentes, o tempo de processamento pode ser elevado para conjuntos de dados relativamente grandes, como o caso do estudado. Uma opção que poderia ser testada é o ***MiniBatchKMeans***, variação do K-médias desenvolvida no contexto de *Big Data* que trabalha com partes do conjunto completo, aleatoriamente escolhidas, mas que compartilham da mesma função de convergência.

Referências

[1] G1. *Inadimplência no cartão cresce mesmo após nova regra do rotativo (https://g1.globo.com/economia/seu-dinheiro/noticia/inadimplencia-no-cartao-cresce-mesmo-apos-nova-regra-do-rotativo.ghtml)*, 2017 (acessado 03 de dezembro de 2018).

[2] I-Cheng Yeh and Che-hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473–2480, 2009.

[3] Wei-Yang Lin, Ya-Han Hu, and Chih-Fong Tsai. Machine learning in financial crisis prediction: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):421–436, 2012.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Guilherme de Alencar Barreto. Introdução à clusterização de dados. Slides da disciplina TIP8311 - Reconhecimento de Padrões, 2018.

[6] Guilherme de Alencar Barreto. Introdução à classificação de padrões. Slides da disciplina TIP8311 - Reconhecimento de Padrões, 2018.

[7] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[8] Stephen M Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[9] Max Kuhn and Kjell Johnson. *Applied predictive modeling*, volume 26. Springer, 2013.

[10] Guilherme de Alencar Barreto. Critério map e classificadores gaussianos. Notas de aula sobre classificadores gaussianos, 2014.

[11] In-Kwon Yeo and Richard A Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.