Relatório
Classificação de Arritmias

Felipe Morine Magami[1]

[1]Escola de Artes, Ciências e Humanidades – Universidade de São Paulo (USP)
São Paulo – SP – Brazil

1. Introdução

O objetivo deste trabalho computacional foi o de avaliar o desempenho de dois diferentes modelos de aprendizado de máquina aplicados a um conjunto de dados de eletrocardiograma. Este relatório descreve os fundamentos teóricos que foram necessários para a realização deste trabalho, explica como foi realizado os experimentos, quais seus resultados, e por fim o que podemos concluir deles.

2. Fundamentação teórica

Nesta seção explicaremos sobre os fundamentos e tecnologias que nortearam a realização deste trabalho.

2.1. Desbalanceamento de classes e *sampling*

Um conjunto de dados desbalanceado é aquele cuja proporção de uma ou mais classes é muito maior do que das outras classes, de forma que os modelos de classificação podem acabar ignorando essas classes minoritárias. Este problema é intrínseco a muitos domínios, como em detecção de fraude, classificação de texto, e diagnósticos médicos [Chawla et al. 2004].

Os métodos que visam lidar com o problema de desbalanceamento em geral podem ser divididos em três categorias:

- *oversampling*: técnicas em que novas instâncias das classes minoritárias são criadas/repetidas;
- *undersampling*: técnicas em que se diminiui a proporção das classes majoritárias, seja eliminando instâncias ou criando novas instâncias dessa classe mas em uma menor proporção;
- técnicas que combinam as duas abordagens acima.

Entre as técnicas de *undersampling*, uma das mais simples é a de *undersampling* randômico. Nela, escolhe-se aleatoriamente um número de amostras da classe majoritária, em geral um número próximo da proporção da classe minoritária, em um problema de classificação binária, e se descarta o resto. Similarmente *oversampling* randômico escolhe aleatoriamente um número de instâncias da classe minoritária e as replica. O ponto fraco de *undersampling* randômico é que as instâncias eliminadas podem levar consigo muita informação importante, e *oversampling* randômicos, por apresentarem várias instâncias copiadas, podem levar à um overfitting. Dito isso, por serem métodos rápidos e diretos, podem ser boas primeiras tentativas para se solucionar um problema de desbalanceamento de classes [Prati et al. 2008].

2.2. PCA

O PCA (*Principal Component Analysis*, Análise de Componentes Principais) é uma técnica aplicada principalmente como método de redução de dimensionalidade de um conjunto de dados, de forma que o novo conjunto reduzido preserve o máximo de variabilidade possível, realizando uma transformação linear do espaço de dimensões original para um novo, e descartando as dimensões menos "informativas" neste último [Jolliffe and Cadima 2016].

Resumidamente, o PCA funciona da seguinte forma: dada uma matriz $X_{n\times d}$, sendo X neste caso o nosso conjunto de dados com n instâncias e d características numéricas, sem o vetor de classes, é calculado a matriz de covariância (ou de correlação, para dados normalizados) $S_{d\times d}$ da matriz X. É necessário então, encontrar λ e $\boldsymbol{a}$ que satisfazem a equação:

$$S\boldsymbol{a} = \lambda\boldsymbol{a} \tag{1}$$

sendo que existem d vetores $\boldsymbol{a}$ e d valores λ (possivelmente não distinctos) que satisfazem a equação acima. Estes são chamados de autovetores e autovalores de S, respectivamente.

O novo conjunto de dados $X^*_{n\times k}$ é a matrix cuja cada coluna $\boldsymbol{p_c}$ é:

$$\boldsymbol{p_c} = X\boldsymbol{a_c} \tag{2}$$

tal que $\boldsymbol{a_c}, c = 1, ..., k$ são os k autovetores correspondentes aos k maiores autovalores $\lambda_c, c = 1, .., k$, ordenados de forma decrescente. Estes $\boldsymbol{p_c}$ são chamados de componentes principais.

Embora como definir o k ideal seja uma questão em aberto, abordagens como escolher um k cuja diferença entre λ_k e λ_{k-1} seja grande [Theodoridis and Koutroumbas 2008], ou escolher os k autovetores $\boldsymbol{a}$ cuja variância explicada combinada seja maior que um limiar (comumente 70% do total) [Jolliffe and Cadima 2016] aparecem na literatura. O PCA também pode ser utilizado como ferramenta da visualização de dados de alta dimensão. Neste caso, k geralmente é definido como 2 ou 3, por razões óbvias.

2.3. KNN

O KNN (k *Nearest Neighbors, k vizinhos mais próximos* é um método de aprendizado baseado em instância cujo bias indutivo se apoia nos conceitos de analogia e similaridade. Ele assume que, dado uma amostra $\bar{\boldsymbol{x}}$ cuja classe é desconhecida, a sua classe prevista deve ser a mesma das k classes mais próximas [Daumé III 2013]. O treinamento do KNN é apenas o armazenamento do conjunto de dados com o vetor de classe correspondente.

Para tentar predizer a classe de uma instância desconhecida, ele calcula a distância entre a instância desconhecida e os outros pontos do conjunto de dados classificado, e classifica a instância desconhecida com a classe que tem mais "votos" dentre as k instâncias mais próximas dele. Por isso ele também é conhecido como um método de aprendizado *lazy* ("preguiçoso"), pois ele demora mais para classificar instâncias inéditas do que na etapa de treinamento [Mitchell 1997].

O algoritmo 1 detalha as etapas descritas anteriormente.

Algorithm 1 KNN

procedure KNN-TRAIN($X, \boldsymbol{y}, k$)
    armazenar X, $\boldsymbol{y}$ e k
    return
procedure KNN-PREDICT($\bar{\boldsymbol{x}}$)
    $D \leftarrow []$ ▷ lista de distâncias
    for $\boldsymbol{x}$ in X **do** ▷ cada instância de X
        $distance \leftarrow calculate\_distance(\boldsymbol{x}, \bar{\boldsymbol{x}})$
        $D.insert(< \boldsymbol{x}, distance >)$
    $D.sort\_list\_by\_distance()$
    $C \leftarrow []$ ▷ lista de classes dos vetores mais próximos
    for $i \leftarrow 1$ to k **do**
        $< \boldsymbol{x}_i, distance_i > \leftarrow D[i]$
        $C.insert(\boldsymbol{y}[\boldsymbol{x}_i])$ ▷ insere em D a classe correspondente y a x
    return $most\_frequent\_class(C)$ ▷ A classe mais "votada"

Existem várias funções de cálculo de distância. A mais utilizada é provavelmente a distância Euclidiana. Dado dois vetores $\boldsymbol{x}$ e $\bar{\boldsymbol{x}}$, com $\boldsymbol{x} = (x_1, ..., x_d)^T$ e $\bar{\boldsymbol{x}} = (\bar{x_1}, ..., \bar{x_d})^T$, a distância Euclidiana entre eles é dada pela equação:

$$Distance(\boldsymbol{x}, \bar{\boldsymbol{x}}) = \sqrt{\sum_{i=1}^{d} (x_i - \bar{x_i})^2} \tag{3}$$

Existem algumas variações do KNN. Uma delas substitui o parâmetro k por um novo parâmetro c, tal que neste novo modelo são coletados os "votos" das instâncias que estão dentro da hiperesfera centrada na instância que se deseja classificar de raio c. Outra modificação é a de se ponderar os 'votos" pela distância encontrada entre a instância desconhecida e seus k vizinhos. Essas abordagens tentam mitigar o problema que pode surgir quando um k inadequado para a predição da instância desconhecida pode incluir 'votos" de instâncias que estão suficientemente longe, e potencialmente o classificando de forma errada [Daumé III 2013]. Outros problemas incluem:

- a sensibilidade do KNN às escalas das dimensões do conjunto de dados;
- o custo de se classificar uma instância inédita para cojuntos de dados de treinamento muito grandes;
- a sensibilidade do KNN à maldição da dimensionalidade. Como o cálculo de distâncias avalia de forma semelhante cada uma das características, características que contribuem pouco ao problema podem fazer com que as distâncias sejam "distorcidas".

2.4. MLP

A MLP (*Multilayer Perceptron*, *Perceptron multicamadas*) é provavelmente a mais famosa rede neural artificial de múltiplas camadas. O seu desenvolvimento teve inspiração nas redes neurais biológicas, que são redes com unidades relativamente simples altamente interconectadas. Uma MLP é formada por várias camadas que contém um certo número

de unidades básicas chamadas *Perceptrons*[Mitchell 1997]. Estes serão explicados mais detalhadamente.

O *Perceptron* funciona da seguinte forma: ele calcula uma combinação linear de suas entradas e um vetor de pesos, e utiliza esta combinação como entrada para uma função, a chamada *função de ativação*. A saída deste neurônio *Perceptron*, portanto, é a saída desta função de ativação.

A figura 1 ilustra um *Perceptron*. Cada $x_1, ..., x_n$ é uma entrada da unidade, que deve ser combinada com o seu respectivo peso $w_1 ... w_n$. Também é utilizado um termo de bias, que pode ser interpretado como um termo que desloca o hiperplano. Este termo é representado pelo peso w_0 e pela entrada extra x_0, que sempre será igual a 1. O resultado, então, é passado para uma função de ativação, que calculará a saída do neurônio. Neste exemplo, a função de ativação é uma função *threshold*, que produz saída 1 caso a somatória seja maior que 0, caso contrário, produz a saída -1.

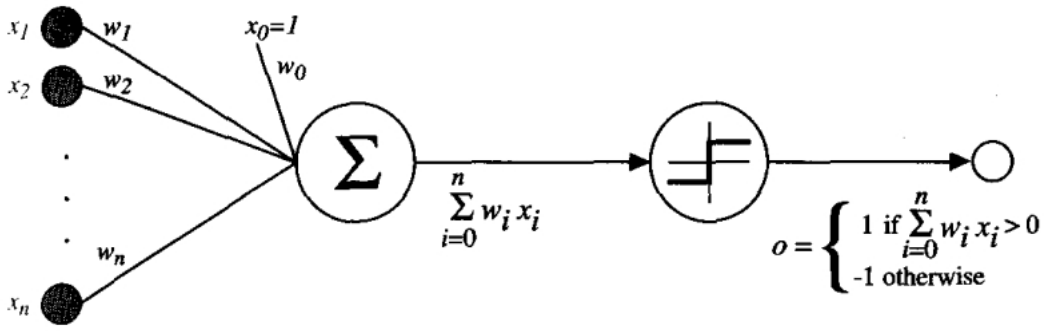

Figura 1. Exemplo de um *Perceptron* [Mitchell 1997]

Uma função de ativação alternativa mais utilizada é a função sigmoide:

$$\sigma(x) = \frac{1}{1 + exp(-x)} \tag{4}$$

onde $exp(\cdot)$ representa a função exponencial. A derivada da função sigmoide é:

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x) \tag{5}$$

Uma MLP, como foi dito antes, é uma rede de várias camadas com vários *Perceptrons*, e cujas saídas dos neurônios das camadas anteriores são usadas como entrada para a camada logo à frente. Em sua versão mais simples, uma MLP consiste em uma camada de entrada, em que cada neurônio é apenas cada x_i do vetor de entrada $\boldsymbol{x} = (x_1, ..., x_D)^T$, uma camada escondida, cujo número de neurônios é determinado pelo projetista do modelo, e uma camada de saída, geralmente com o número de neurônios igual ao número de classes diferentes para um problema de classificação.

A figura 2 mostra um exemplo de uma MLP com D entradas, uma camada escondida de M neurônios, e uma camada de saída com K neurônios. Cada $w_{ij}^{(1)}$ representa o peso da entrada x_j para o neurônio da camada escondida z_i. Similarmente, cada $w_{ij}^{(2)}$

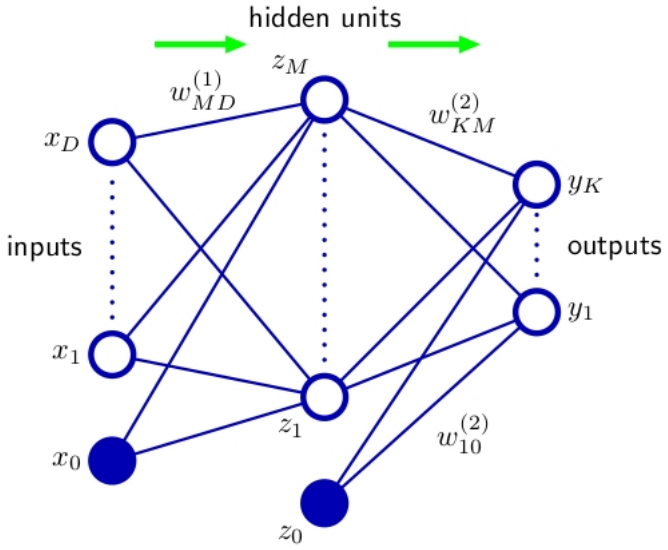

Figura 2. Exemplo de um *MLP* [Bishop 2006]

representa o peso da saída do neurônio escondido z_j para o neurônio da camada de saída y_i. x_0 e z_0 são termos de bias, e como explicados anteriormente, tem valor 1.

As setas verdes na figura 2 estão relacionadas ao processo de *feedfowarding*. É através deste processo que a entrada é "alimentada" para as camadas seguintes. Primeiramente, são calculadas as entradas para cada neurônio na camada escondida. Ou seja, a entrada a_j para o neurônio z_j é dado por [Bishop 2006]:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{6}$$

e a saída do neurônio é dada por:

$$z_j = \sigma(a_j) \tag{7}$$

para uma função de ativação sigmoide na camada escondida.

De forma similar, a entrada b_k para o neurônio de saída y_k é dada por:

$$b_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \tag{8}$$

e a saída utilizando uma função sigmoidal:

$$y_k = \sigma(b_k) \tag{9}$$

Podemos, então, combiná-las, de forma que a saída y_k da rede após o processo de feedfowarding, seja:

$$y_k(\boldsymbol{x}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} \sigma\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right) \tag{10}$$

O principal algoritmo de treinamento de uma MLP é o algoritmo de *backpropagation*. Ele tem esse nome porque sua ideia é a de se "propagar" os erros da camada de saída (que podemos calcular por ser um aprendizado supervisionado) pelos pesos da rede, realizando o que se assemelha a um caminho inverso do método de *feedfowarding*. O *backpropagation* se baseia no algoritmo de gradiente descendente para a minimização de funções, em que os parâmetros a serem atualizados são os pesos $w_{ij}^{(l)}$. Explicaremos detalhadamente o algoritmo para uma função de erro de mínimos quadrados médios, que tem a forma [Duda et al. 2012]:

$$J_T = \frac{1}{N}\sum_{n=1}^{N} J(\boldsymbol{x}_n) \tag{11}$$

em que $\boldsymbol{x}_n$ é uma instância do conjunto com N instâncias e

$$J(\boldsymbol{x}_n) = \frac{1}{2}\sum_{k=1}^{K} \left(t_k(\boldsymbol{x}_n) - y_k(\boldsymbol{x}_n)\right)^2 \tag{12}$$

onde $t_k(\boldsymbol{x}_n)$ é a saída esperada para a saída y_k ao utilizamos como entrada a instância $\boldsymbol{x}_n$.

Como no algoritmo de gradiente descendente em "batelada", os pesos são atualizados pela seguinte fórmula:

$$w_{ij}^{(l)}(m+1) = w_{ij}^{(l)}(m) + \Delta w_{ij}^{(l)}(m) \tag{13}$$

em que m indica a iteração do algoritmo, e $\Delta w_{ij}^{(1)}$ é:

$$\Delta w_{ij}^{(l)} = -\eta \frac{\partial J_T}{\partial w_{ij}^{(l)}} == -\eta \frac{1}{N}\sum_{n=1}^{N} \frac{\partial J(\boldsymbol{x}_n)}{\partial w_{ij}^{(l)}} \tag{14}$$

sendo que $-\frac{\partial J}{\partial w_{ij}^{(l)}}$ é o gradiente negativado, que indica a "direção" que em termos de $w_{ij}^{(l)}$ minimiza a função J, e η é a chamada *taxa de aprendizado*, um escalar que define o tamanho do "passo" que deve se dar nessa direção, geralmente definido pelo projetista do modelo. Um η muito grande pode fazer com que o modelo não consiga convergir, enquanto um passo muito pequeno pode fazer com que a convergência seja demorada demais.

Utilizando a regra da cadeia, considerando os pesos entre a camada escondida e a camada de saída para uma rede de apenas uma camada escondida, e função de ativação sigmoidal:

$$-\frac{\partial J}{\partial w_{ij}^{(2)}} = -\frac{\partial J}{\partial b_i}\frac{\partial b_i}{\partial w_{ij}^{(2)}} = \delta_i^{(2)}\frac{\partial b_i}{\partial w_{ij}^{(2)}} \tag{15}$$

$$\frac{\partial b_i}{\partial w_{ij}^{(2)}} = z_j \tag{16}$$

$$\delta_i^{(2)} = -\frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial b_i} = (t_i - y_i)y_i(1-y_i) \tag{17}$$

E para um peso na camada escondida $w_{ij}^{(1)}$:

$$-\frac{\partial J}{\partial w_{ij}^{(1)}} = -\frac{\partial J}{\partial z_i}\frac{\partial z_i}{\partial a_i}\frac{\partial a_i}{\partial w_{ij}^{(1)}} = \delta_i^{(1)}\frac{\partial a_i}{\partial w_{ij}^{(1)}} \tag{18}$$

$$\begin{aligned}
-\frac{\partial J}{\partial z_i} &= -\frac{\partial}{\partial z_i}\left(\frac{1}{2}\sum_{k=1}^{K}(t_k - y_k)^2\right) \\
&= \sum_{k=1}^{K}(t_k - y_k)\frac{\partial y_k}{\partial z_i} \\
&= \sum_{k=1}^{K}(t_k - y_k)\frac{\partial y_k}{\partial b_k}\frac{\partial b_k}{\partial z_i} \qquad (19) \\
&= \sum_{k=1}^{K}(t_k - y_k)y_k(1 - y_k)w_{ki}^{(2)} \\
&= \sum_{k=1}^{K}\delta_k^{(2)}w_{ki}^{(2)}
\end{aligned}$$

$$\frac{\partial z_i}{\partial a_i} = z_i(1 - z_i) \tag{20}$$

$$\frac{\partial a_i}{\partial w_{ij}^{(1)}} = x_j \tag{21}$$

$$\delta_i^{(1)} = z_i(1 - z_i)\sum_{k=1}^{K}\delta_k^{(2)}w_{ki}^{(2)} \tag{22}$$

Finalmente, podemos definir $\Delta w_{ij}^{(l)}$ tanto para os pesos que partem da camada de entrada para a camada escondida $l = 1$ e os pesos $l = 2$ que saem da camada escondida para a camada de saída. Uilizando as informações de $\delta_i^{(1)}$ e $\delta_i^{(2)}$, encontrados nas equações 22 e 17:

$$\Delta w_{ij}^{(1)} = \eta\frac{1}{N}\sum_{n=1}^{N}\delta^{(1)_i}(\boldsymbol{x}_n)x_j \tag{23}$$

$$\Delta w_{ij}^{(2)} = \eta\frac{1}{N}\sum_{n=1}^{N}\delta^{(2)_i}(\boldsymbol{x}_n)z_j \tag{24}$$

O algoritmo 2 *Backpropagation*, por fim combina todas essas informações para se treinar uma rede neural. Dado um conjunto de dados X com seu respectivo vetor de classes $\boldsymbol{y}$, uma taxa de aprendizado η e um número de neurônios da camada escondida n_{hidden} [Mitchell 1997]:

O processo de predição de uma MLP para uma instância inédita é apenas a etapa de *feedfowarding* para uma rede previamente treinada. Para problemas de classificação, a classe predita é aquela que remete ao neurônio de saída com o maior valor (por isso um número de neurônios igual à quantidade de diferentes classes, embora um problema

Algorithm 2 Backpropagation

procedure BACKPROPAGATION($X, \boldsymbol{y}, \eta, n_{hidden}$)
    construir a rede, com D neurônios na camada de entrada, n_{hidden} neurônios na camada escondida, e K neurônios de saída, e seus respectivos pesos $w_{ij}^{(l)}$
    atribuir um pequeno valor randômico, como entre 0.5 e -0.5, aos pesos
    while critério de parada não alcançado **do**
        for Cada par $< \boldsymbol{x}, y >$ in $X, \boldsymbol{y}$ **do**
            alimentar a rede com $\boldsymbol{x}$ utilizando o método de *feedfowarding*
                                                                ▷ *backpropagate*
            calcular todos os $\delta_i^{(l)}$ utilizando as equações 22 e 17
            calcular todos os $\Delta w_{ij}^{(l)}$utilizando as equações 23 e 24
            atualizar os pesos utilizando a equação 13

binário poderia, dependendo da função de ativação da camada de saída, utilizar apenas um neurônio de saída).

Alguns dos parâmetros que devem ser definidos ao se construir uma MLP são:

- o número de camadas escondidas. Foi provado que uma MLP com apenas uma camada escondida é o suficiente para se modelar qualquer espaço de decisão, mas não é claro o número de neurônios escondidos a ser utilizado [Duda et al. 2012];
- o número de neurônios em cada camada escondida. Poucos neurônios podem não fornecer à rede uma capacidade suficiente de generalização, mas muitos neurônios podem levar ao *overfitting*[Duda et al. 2012];
- a taxa de aprendizado;
- o critério de parada. Alguns deles são:
  - um número de iterações (épocas) máximo;
  - um valor mínimo para a norma do gradiente do erro;
  - quando o erro de classificação do conjunto de treinamento não diminui por algumas épocas

Existe uma infinidade de possíveis modificações para redes neurais. Elas vão desde modificações mais simples, como a inserção de termos de regularização para tentar mitigar o *overfitting* e termos de momento para tentar fazer com que a convergência do algoritmo não seja em um mínimo local, a mudanças nas funções de erro e no algoritmo de treinamento, a até modificações na própria estrutura da rede, como as redes recorrentes, em que as saídas em um *feedfowarding* de um neurônio serve como entrada em uma iteração seguinte [Mitchell 1997].

As redes neurais são bastante vulneráveis à *overfitting*. Algumas técnicas para se tentar mitigar esse problema já foram citadas, como termos de regularização ou critérios de parada que não esperam o modelo convergir [Duda et al. 2012].

3. Experimento

O conjunto de dados escolhido para esse trabalho foi a base de dados MIT-BIH Arrythmia Database[Moody and Mark 2001][Goldberger et al. 2000], composto por gravações de eletrocardiogramas de 47 indivíduos, cujo maior interesse era o de disponibilizar um

conjunto de dados comum procurando acelerar as pesquisas voltadas para análise de arritmias. No nosso trabalho, utilizamos um conjunto de dados com características previamente extraídas, disponibilizadas pelo orientador da disciplina.

O conjunto de dados completo contém 103579 instâncias e 7 características mais um vetor contendo 15 classes, sendo elas: $\mathbf{1}$, $\mathbf{2}$, $\mathbf{3}$, $\mathbf{5}$, $\mathbf{6}$, $\mathbf{7}$, $\mathbf{8}$, $\mathbf{9}$, $\mathbf{11}$, $\mathbf{12}$, $\mathbf{13}$, $\mathbf{15}$, $\mathbf{16}$, $\mathbf{17}$ e $\mathbf{18}$ (note que alguns "valores" no intervalo $[\mathbf{1},\mathbf{15}]$ estão faltando). Este conjunto de dados foi primeiramente separado em um conjunto de treinamento e um conjunto de teste, de forma randômica e estratificada, em uma proporção de 30% das instâncias para o conjunto de teste e 70% para o conjunto de treinamento, e aplicado uma normalização para ambos usando os valores do conjunto de treinamento.

A figura 3 mostra a distribuição das classes para o conjunto de treinamento. De fato, a classe majoritária $\mathbf{1}$ é representada por 50221 instâncias, mais do que 69% do conjunto, a segunda classe majoritária $\mathbf{2}$ representa 7.6% do total, e no outro extremo, temos a classe $\mathbf{8}$, com apenas uma instância.

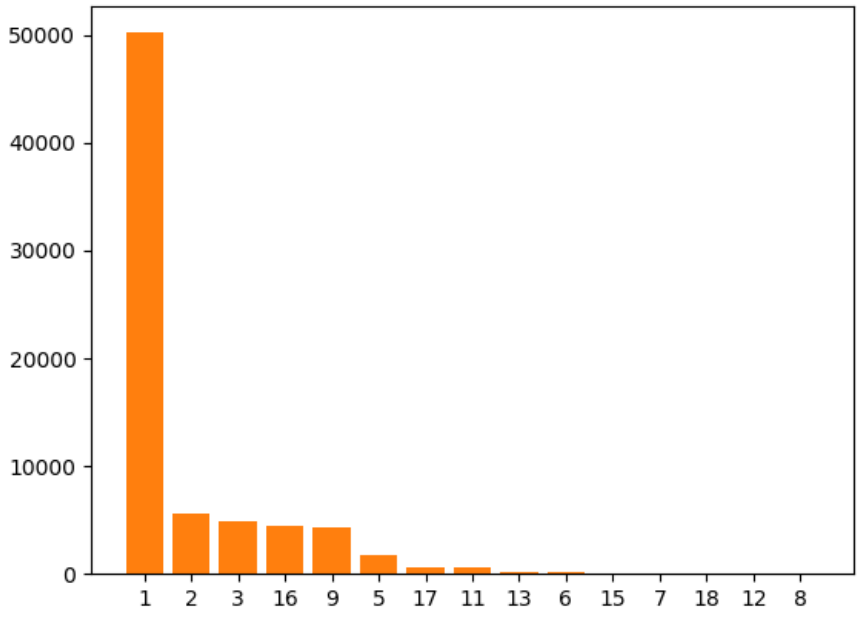

Figura 3. Distribuição das classes para o conjunto de treinamento completo.

Nitidamente percebe-se que poderemos ter problemas com desbalanceamento das classes. Foi criado, então, mais um conjunto de dados, em que se aplicou *undersampling* randômico na classe majoritária $\mathbf{1}$, reduzindo seu número para 7000, número mais próximo da classe $\mathbf{2}$. A figura 3 mostra a distribuição para este conjunto.

Foi considerado utilizar alguma técnica de *oversampling* para tentarmos balancear as classes minoritárias, mas essa ideia foi descartada pelas potenciais distorções que o dataset pode sofrer, especialmente para o caso da classe com apenas uma instância $\mathbf{8}$.

Finalmente, uma transformação via PCA foi aplicada para ambos os conjuntos

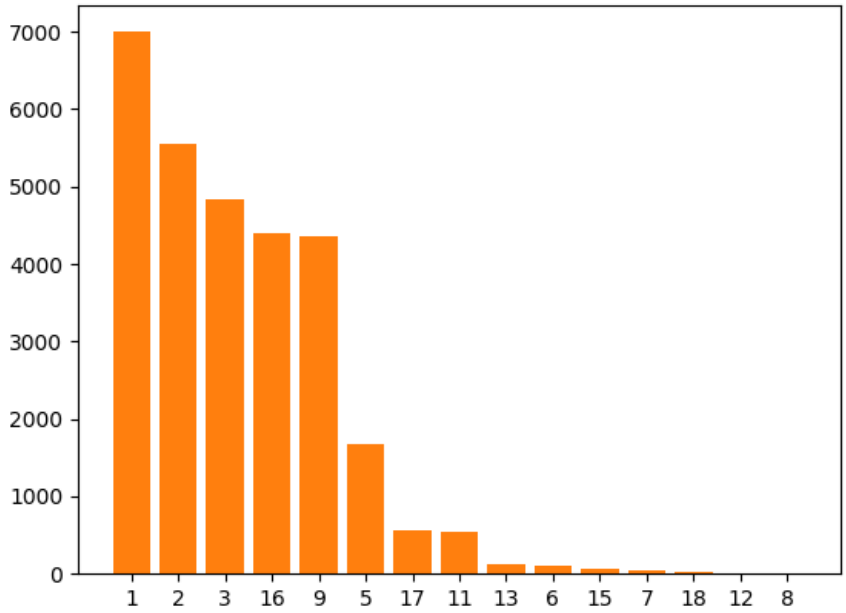

Figura 4. Distribuição das classes para o conjunto de treinamento reduzido.

de treinamento completo e o reduzido via *undersampling* (e obviamente o de teste, já o que espaço de dimensões foi alterado), utilizando os autovetores e autovalores encontrados para o conjunto de treinamento completo, e escolhido as 4 componentes principais que expressam maior variância para compor outros dois novos conjuntos de dados, um completo com PCA e outro reduzido por *undersampling* randômico com PCA. A tabela 3 mostra a porcentagem de variância explicada por cada uma das componentes principais escolhidas.

pc_1	pc_2	pc_3	pc_4	Total
25.86%	16.58%	14.29%	14.11%	70.34%

Tabela 1. Variância explicada das componentes principais.

Em resumo, foram criados quatro conjuntos de dados de treinamento: um completo, em que nada foi aplicado; um com todas as instâncias (exceto, obviamente as do conjunto de teste), em que foi aplicado transformação PCA para quatro componentes principais; um cujas instâncias foram reduzidas utilizando um método de *undersampling* randômico, e por fim um em que foi aplicado transformação PCA para o conjunto reduzido. Obtemos também dois conjuntos de teste, um com PCA aplicado e outro sem.

Para um dos quatro conjuntos de treinamento foram aplicados seis diferentes classificadores, todos de implementação própria:

- três KNNs, para $k = 3, 4, 5$;
- três MLPs de uma camada escondida, com funções sigmoide tanto para a camada escondida quanto para a camada de saída, treinadas por 6000 épocas e com taxa de aprendizado $\eta = 0.6$.

Uma vez treinados, foi feita uma avaliação de acurácia e matriz de confusão utilizando o conjunto de teste apropriado e método holdout.

4. Resultados e discussão

Nesta seção serão apresentados os resultados. Para cada conjunto de treinamento, será mostrado a acurácia dos seis classificadores, e a matriz de confusão para os modelo de KNN e MLP que obtiverem a melhor acurácia[1]. As porcentagens são as proporções classificadas para uma classe y cuja saída prevista é t, o que significa que os valores na diagonal da matriz são as taxas de recall para a classe, e as cores refletem essa porcentagem (quanto mais escuras, maior essa porcentagem).

4.1. Conjunto completo sem PCA

A tabela 2 mostra a taxa de acurácia para os modelos treinados no conjunto de dados completo sem aplicação do PCA. As figuras 5 e 6 mostram as matrizes de confusão para os modelos diferentes que obtiveram melhor acurácia (KNN com $k = 5$ e MLP com $n_{hidden} = 9$)

Modelo	k ou n_{hidden}	% Acurácia
KNN	3	82.06%
KNN	4	82.43%
KNN	**5**	**83.1%**
MLP	3	75.28%
MLP	6	75.63%
MLP	**9**	**75.67%**

Tabela 2. Acurácia para o conjunto de treinamento completo sem PCA

[1]As matrizes não mostradas neste relatório estarão disponibilizadas juntamente com os códigos e outros documentos produzidos neste experimento

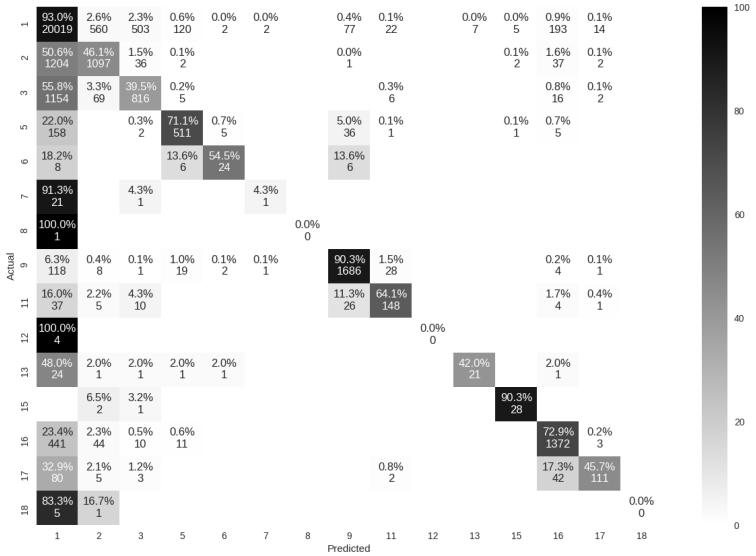

Figura 5. Matriz de confusão para KNN com $k = 5$ sobre o conjunto de treinamento completo sem PCA.

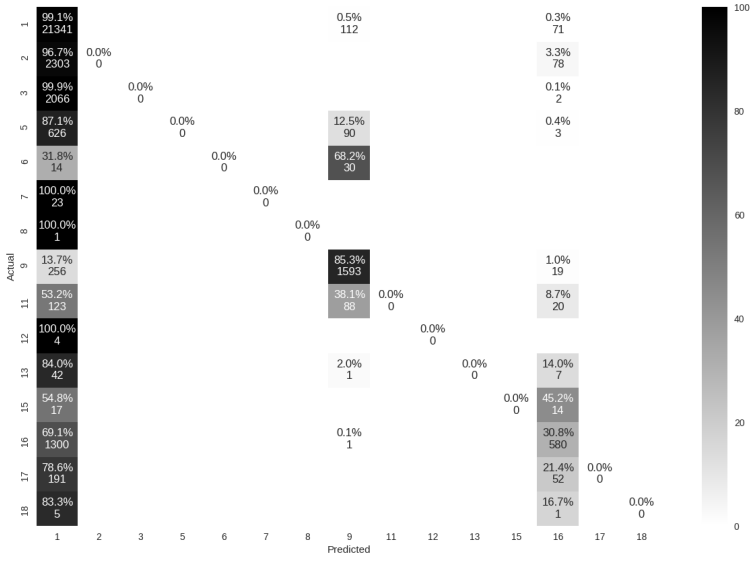

Figura 6. Matriz de confusão para MLP com $n_{hidden} = 9$ sobre o conjunto de treinamento completo sem PCA.

O KNN teve uma performance surpreendentemente alta. Embora a classificação tenha sido enviesada para a classe majoritária **1**, em que mesmo para classes com uma proporção maior de instâncias (como **2** e **3**) tiveram mais de 50% de suas instâncias de teste classificadas erroneamente como sendo da classe **1**, o KNN também obteve um bom

desempenho para as classes $\mathbf{9}$ e $\mathbf{15}$. De fato, as únicas classes em que nenhuma instância foi classificada corretamente foram as três classes minoritárias (e tendo em mente o funcionamento do KNN e o fato de existir apenas uma instância de treinamento da classe $\mathbf{8}$, fica claro que nenhum valor k testado irá fazer com que o modelo classifique a instância de teste de classe $\mathbf{1}$ corretamente).

Já no caso da MLP, percebe-se que a maior parte da sua taxa de acurácia é graças à alta classificação correta de instâncias de classe $\mathbf{1}$. Neste caso, a MLP tendeu a errar as outras classes, em que a rede previu apenas três classes diferentes para as 15 possíveis, $\mathbf{1}$, $\mathbf{9}$ e $\mathbf{16}$.

4.2. Conjunto completo com PCA

A tabela 3 mostra a taxa de acurácia para os modelos treinados para o conjunto de treinamento completo com aplicação do PCA. As figuras 7 e 8 mostram as matrizes de confusão para os modelos diferentes que obtiveram melhor acurácia (KNN com $k = 5$ e MLP com $n_{hidden} = 9$)

Modelo	k ou n_{hidden}	% Acurácia
KNN	3	73.54%
KNN	4	74.69%
KNN	**5**	**76.00%**
MLP	3	73.51%
MLP	6	73.62%
MLP	**9**	**73.81%**

Tabela 3. Acurácia para o conjunto de treinamento completo com PCA

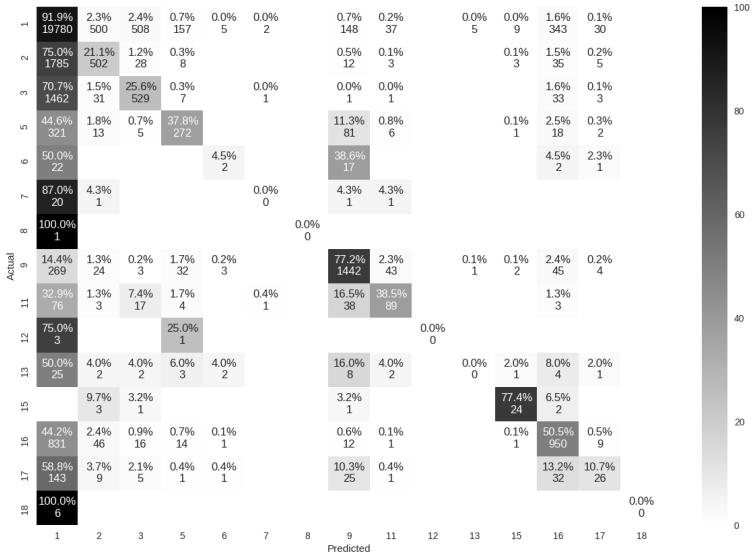

Figura 7. Matriz de confusão para KNN com $k = 5$ sobre o conjunto de treinamento completo com PCA.

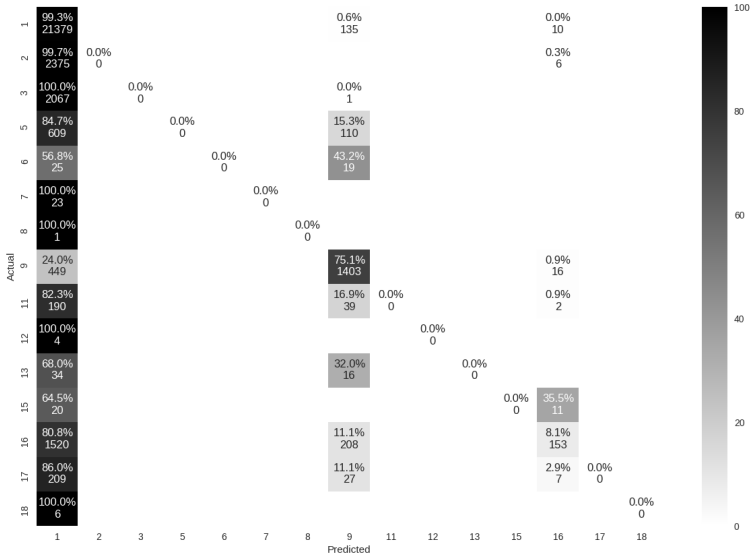

Figura 8. Matriz de confusão para MLP com $n_{hidden} = 9$ sobre o conjunto de treinamento completo com PCA.

O KNN teve a performance bastante afetada pela aplicação do PCA. O recall de quase todas as classes caíram, com atenção especial à classe **13**, que não classificou nenhuma de suas instâncias corretamente, comparado com 21 para a predição sem PCA.

Já a MLP não mostrou uma diferença significativa se comparado com o treina-

mento sem PCA, com uma taxa de acurácia 2% menor. A MLP continua classificando a maioria das instâncias de teste como sendo da classe majoritária **1**. O modelo, como o anterior, também parece ter a capacidade de predizer apenas as três classes mencionadas anteriormente.

4.3. Conjunto reduzido sem PCA

A tabela 4 mostra a taxa de acurácia para os modelos treinados para o conjunto de treinamento reduzido por *undersampling* randômico sem aplicação do PCA. As figuras 9 e 10 mostram as matrizes de confusão para os modelos diferentes que obtiveram melhor acurácia (KNN com $k = 5$ e MLP com $n_{hidden} = 3$)

Modelo	k ou n_{hidden}	% Acurácia
KNN	3	65.18%
KNN	4	66.08%
KNN	**5**	**66.52%**
MLP	**3**	**74.21%**
MLP	*6*	*46.82%*
MLP	3	54.00%

Tabela 4. Acurácia para o conjunto de treinamento reduzido sem PCA

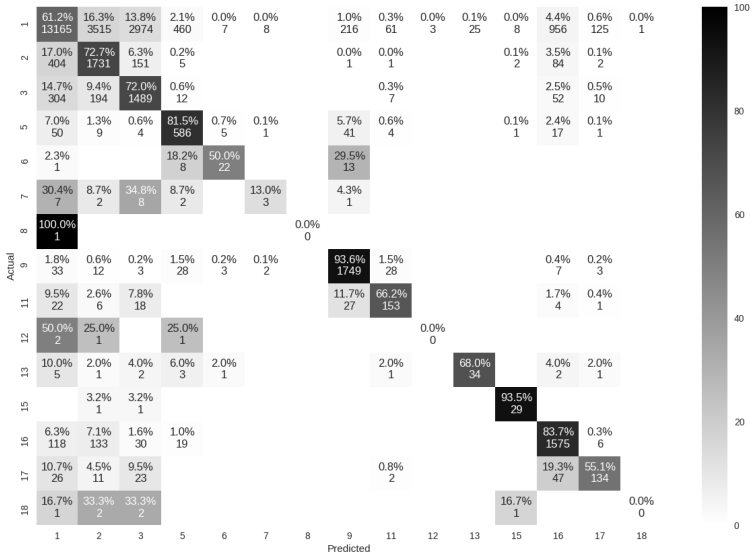

Figura 9. Matriz de confusão para KNN com $k = 5$ sobre o conjunto de treinamento reduzido sem PCA.

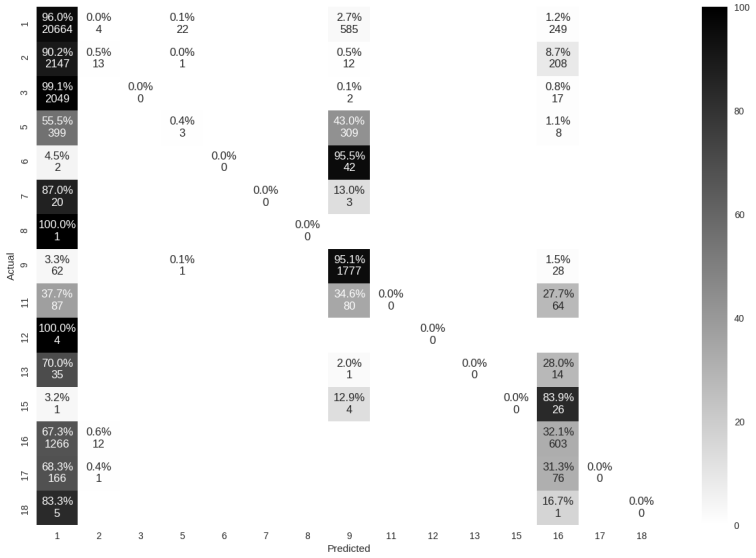

Figura 10. Matriz de confusão para MLP com $n_{hidden} = 3$ sobre o conjunto de treinamento reduzido sem PCA.

Para as classes $\mathbf{2}$ até $\mathbf{18}$, o KNN com conjunto de dados reduzido mostrou uma performance similar à do melhor KNN para o conjunto completo sem PCA, com melhorias significativas para o recall das classes $\mathbf{2}$, $\mathbf{3}$, $\mathbf{5}$ e $\mathbf{16}$. Porém, é notável que o classificador errou muito mais a classe majoritária $\mathbf{1}$, o que se refletiu na taxa de acurácia baixa.

Para os classificadores MLP, os resultados foram variados. A taxa de acurácia do melhor modelo ($n_{hidden} = 3$) foi bem similar às dos modelos MLP anteriores. Uma pequena diferença é que algumas instâncias foram classificadas como sendo das classes $\mathbf{2}$ e $\mathbf{5}$, embora o recall dessas classes ainda é extremamente baixo. Uma tendência que observamos nas duas primeiras matrizes de confusão das MLPs e neste modelo está bem mais presente é que o modelo tende a confundir instâncias da classe $\mathbf{6}$ como sendo da classe $\mathbf{9}$, e instâncias da classe $\mathbf{15}$ como sendo da classe $\mathbf{16}$, o que reduziu a precisão dessas classes preditas.

Contudo, observamos que obtivemos um modelo MLP, com seis neurônios na camada escondida, que produziu uma taxa de acurácia de apenas 46.82%, o mais baixo de todos os modelos treinados. A figura 11 mostra a matriz de confusão para este modelo.

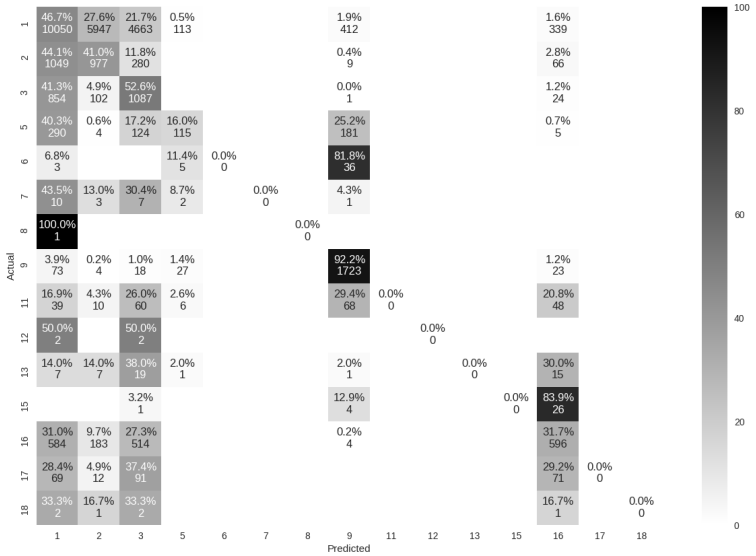

Figura 11. Matriz de confusão para MLP com $n_{hidden} = 6$ sobre o conjunto de treinamento reduzido sem PCA.

O principal motivo para a baixa taxa de acurácia é a baixa performance do modelo para a classe majoritária $\mathbf{1}$., com um recall de 46%. Se comparado ao modelo com três camadas escondidas, podemos observar que existe um menor viés para a classe $\mathbf{1}$, e de fato, o modelo conseguiu taxas de recall muito mais altas para as classes $\mathbf{2}$, $\mathbf{3}$ e $\mathbf{5}$. Contudo, se comparado ao resultado do modelo KNN treinado com este mesmo conjunto de dados, os resultados são em geral muito piores para todas as classes.

4.4. Conjunto reduzido com PCA

A tabela 5 mostra a taxa de acurácia para os modelos treinados para o conjunto de treinamento reduzido por *undersampling* randômico com aplicação do PCA. As figuras 9 e 10 mostram as matrizes de confusão para os modelos diferentes que obtiveram melhor acurácia (KNN com $k = 5$ e MLP com $n_{hidden} = 3$)

Modelo	k ou n_{hidden}	% Acurácia
KNN	3	50.84%
KNN	4	52.28%
KNN	**5**	**53.34%**
MLP	**3**	**59.04%**
MLP	6	58.62%
MLP	9	57.28%

Tabela 5. Acurácia para o conjunto de treinamento reduzido com PCA

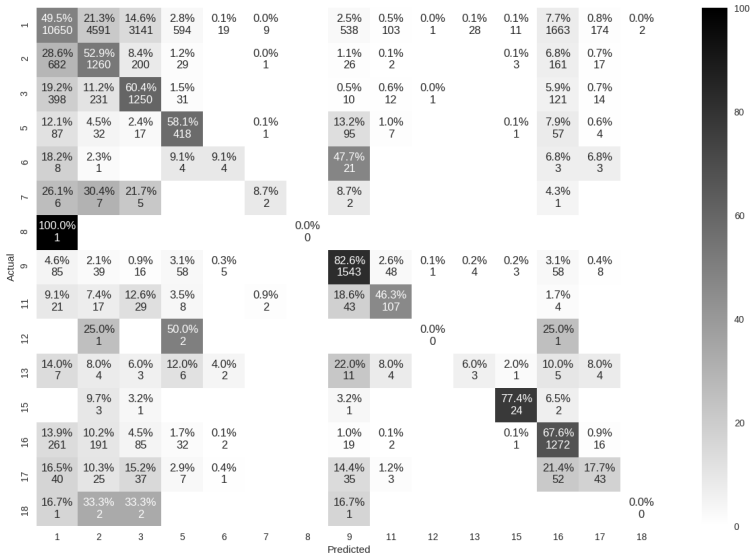

Figura 12. Matriz de confusão para KNN com $k = 5$ sobre o conjunto de treinamento reduzido com PCA.

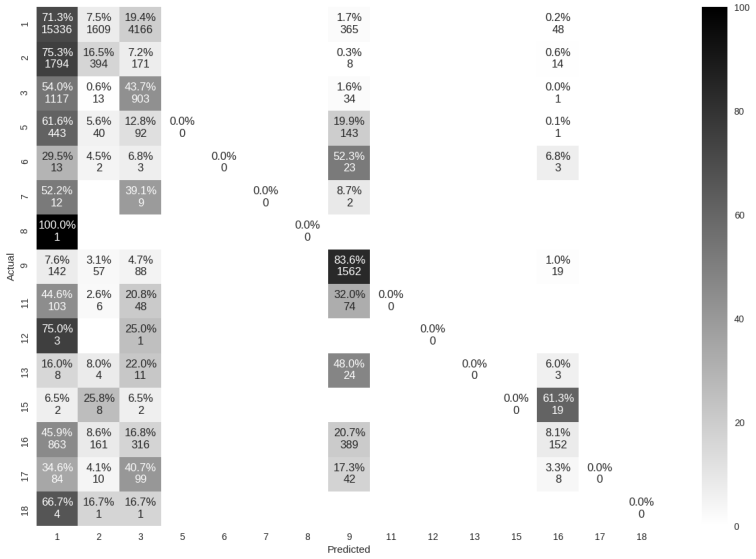

Figura 13. Matriz de confusão para MLP com $n_{hidden} = 3$ sobre o conjunto de treinamento reduzido com PCA.

Apenas para o treinamento dos modelos envolvendo este dataset os modelos MLP tiveram uma acurácia melhor do que as dos modelos KNN. Mais uma vez, o PCA foi prejudicial para a performance dos modelos KNN, cujos resultados foram inferiores se comparados ao modelo para o dataset reduzido sem PCA.

Para o modelo MLP com maior taxa de acurácia, os resultados foram mistos. De fato, o resultado apresentado parece uma mistura dos resultados do melhor modelo MLP para o conjunto reduzido sem PCA e o pior modelo, em que existe um menor enviesamento para a classe $\mathbf{5}$ em troca de um recall um pouco reduzido para esta classe, e um recall maior para as outras. Dito isso, sua performance ainda não foi boa, com recall para as classes $\mathbf{2}$ a $\mathbf{18}$ pior do que os melhores modelos de KNN.

5. Considerações finais e conclusão

Para o conjunto de dados de eletrocardiograma fornecido, o KNN parece ser um modelo superior ao MLP. O modelo de melhor performance foi aquele treinado com o conjunto de treinamento completo e sem aplicação de PCA, com uma acurácia de 83.1%. Contudo, é importante notar que a performance para as classes que não a majoritária foi melhor para o modelo KNN treinado com o conjunto reduzido por *undersampling* sem PCA, o que talvez seja mais interessante mesmo com uma taxa de acurácia mais baixa, de 66.52%.

É importante notar que para todos os conjuntos de treinamento o modelo KNN com melhor performance foi o de $k = 5$, embora os modelos com um k menor tiveram acurácia muito semelhantes. Isto talvez seja um indicativo de que a performance possa ser melhorada aumentando este parâmetro. Contudo, dado a proporção das classes, a performance do modelo pode acabar sendo enviesada para as classes majoritárias, ao ponto de não ser possível a classificação de classes minoritárias dependendo do valor desse parâmetro, como aconteceu com a classe com apenas uma instância. De fato, os baixos valores de k escolhido (em relação ao tamanho do conjunto de treinamento) pode ter ajudado na boa performance dos modelos KNN para algumas classes minoritárias.

O grande desafio do problema foi a distribuição desbalanceada do conjunto de dados. Este pareceu ser a principal razão dos modelos MLP terem tido uma performance inferior, em que as maiores taxas de acurácia se davam apenas pelo fato dos modelos classificarem a maior parte das instâncias do conjunto de teste como sendo da classe majoritária. Uma possível solução, como foi discutida anteriormente, é a de se utilizar técnicas de *oversampling*, para tentarmos balancear a distribuição das classes, mas isso não pode ser adequado, especificamente para as classes minoritárias com apenas dezenas de instâncias ou menos. Uma ideia não relacionada mas que pode ser útil para este problema em específico seria o de unirmos algumas classes em uma só, como foi feito no trabalho de Kandala e Dhuli (2018), embora neste experimento isso não tenha sido permitido.

A aplicação do PCA para os modelos MLP treinados utilizando o conjunto de treinamento completo não pareceu ser significativa, reduzindo a acurácia geral por alguns poucos pontos percentuais. Para o conjunto reduzido, exceto para o modelo com $n_{hidden} = 3$, que estava enviesado para a classe majoritária e por isso apresentou um resultado semelhante aos dos modelos treinados com o conjunto completo, a taxa de acurácia tendeu a aumentar, embora com uma perforamnce ainda abaixo dos modelos KNN sem PCA, em que foi nítido o impacto negativo da aplicação do PCA. É possível que isso se

deu ao que foi discutido na seção 2; uma vez que o KNN considera cada característica igualmente, é possível que a performance dos modelos KNN com aplicação do PCA possa ter sido prejudicada justamente pela diferença na expressividade da variância de cada componente. Neste caso, uma mudança seria colocar pesos para cada componente de acordo com sua variância explicada.

Referências

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

Chawla, N. V., Japkowicz, N., and Kotcz, A. (2004). Special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter*, 6(1):1–6.

Daumé III, H. (2013). *A course in machine learning*. Self Published.

Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., and Stanley, H. E. (2000). Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220.

Jolliffe, I. T. and Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202.

Kandala, R. and Dhuli, R. (2018). Classification of imbalanced ecg beats using re-sampling techniques and adaboost ensemble classifier. *Biomedical Signal Processing and Control*, 41:242–254.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

Moody, G. B. and Mark, R. G. (2001). The impact of the mit-bih arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50.

Prati, R. C., Batista, G. E., and Monard, M. C. (2008). A study with class imbalance and random sampling for a decision tree learning system. In *IFIP International Conference on Artificial Intelligence in Theory and Practice*, pages 131–140. Springer.

Theodoridis, S. and Koutroumbas, K. (2008). *Pattern Recognition, Fourth Edition*. Academic Press, Inc., Orlando, FL, USA, 4th edition.