Capítulo 3

Pré-processamento

Considerando o grande volume de dados disponível em diversas aplicações, com frequência os conjuntos de dados não possuirão uma qualidade boa o suficiente para a extração de conhecimento novo, útil e relevante por algoritmos de aprendizado de máquina (AM). As principais causas de baixa qualidade de dados incluem a ocorrência de atributos irrelevantes, valores ausentes ou redundantes. Neste capítulo serão apresentadas algumas simples abordagens para melhorar a qualidade dos dados, que aumentam as chances de um bom modelo ser induzido por um algoritmo de AM. Na Seção 3.1 será descrito o conjunto de dados que será utilizado para ilustrar o uso de técnicas de pré-processamento. Na Seção 3.2 será apresentado como alguns atributos podem ser removidos manualmente. Na Seção 3.3 serão apresentadas algumas das técnicas de amostragem de dados mais comuns. Na Seção 3.4 será explicado como dados ausentes podem ser tratados. Na Seção 3.5 será abordada a ocorrência de objetos ou atributos redundantes. Por fim, na Seção 3.9 serão propostos alguns exercícios para fixar os conceitos apresentados.

3.1. Conjunto de dados

Para os exemplos apresentados no presente capítulo, serão utilizados dois conjuntos de dados: *Breast Cancer Wisconsin*[1] [Street et al. 1992, Mangasarian et al. 1995] e *Contraceptive Method Choice*[2] [Lim et al. 2000].

No conjunto *Breast Cancer Wisconsin* cada objeto consiste em um tecido de massa mamária e os atributos correspondem a características, extraídas a partir de imagens digitalizadas, dos núcleos celulares (raio, textura, perímetro etc.) contidos em cada tecido. As classes associadas a cada tecido informam o diagnóstico do tecido (maligno ou benigno).

O conjunto *Contraceptive Method Choice* consiste em amostras de uma pesquisa realizada na Indonésia em 1987 sobre métodos contraceptivos. Os objetos consistem em mulheres que não estavam grávidas ou que não sabiam da gravidez ao participarem da pesquisa. Os atributos consistem em características socio-econômicas. O problema deste conjunto de dados consiste em classificar o método contraceptivo utilizado em: 1 (nenhum método utilizado), 2 (método de longa duração) ou 3 (método de curta duração). Todos os atributos são dados qualitativos nominais ou ordinais. Uma descrição completa

[1] `http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29`

[2] `http://archive.ics.uci.edu/ml/datasets/Contraceptive+Method+Choice`

de cada atributo e seus possíveis valores está disponível em `http://archive.ics.uci.edu/ml/machine-learning-databases/cmc/cmc.names`.

Os exemplos apresentados a seguir utilizarão os arquivos `breast_cancer.csv` e `cmc.csv`, fornecidos como material suplementar a este texto.

3.2. Eliminação manual de atributos

Diversos conjuntos de dados do mundo real podem ter atributos que, por serem claramente irrelevantes, não apresentam qualquer benefício para uma tarefa de classificação ou de extração de conhecimento. Por exemplo, o conjunto de dados *Breast Cancer* contém o atributo `sample_id`, um valor numérico que identifica o tecido analisado. Como tal valor será único para cada objeto e o mesmo não possui qualquer sentido comparativo, ele pode ser eliminado. Em Python, isso pode ser realizado por meio do código a seguir.

```
import pandas

dados = pandas.read_csv('breast_cancer.csv')

# removendo a coluna sample_id, a qual nao eh necessaria
# para realizar o diagnostico
del dados['sample_id']
```

3.3. Amostragem

Diversos algoritmos de AM, sejam pela suas complexidades computacionais ou pelas quantidades de memória exigida, apresentam dificuldades em tratar conjuntos de dados de tamanho elevado. Uma quantidade de dados muito grande pode tornar o processo de treinamento demorado. Um meio de contornar essa dificuldade é a utilização de amostras do conjuntos de dados original. A utilização de um subconjunto menor de objetos, em geral tornará mais simples e rápida a geração de um modelo.

Entretanto, um ponto importante a ser levado em consideração está relacionado ao nível ao qual a amostra representa a distribuição original dos dados. Normalmente, em casos nos quais a mesma não seja representativa, o modelo de AM induzido não será capaz de atingir uma eficiência aceitável para o problema.

Um modo para tratar o ponto supracitado, resume-se na utilização de técnicas de amostragem estatística, as quais aumentam as chances de que as amostras extraídas da base de dados original possam ser informativas e representativas. Duas das técnicas mais utilizadas são a amostragem simples e a amostragem estratificada. Ambas serão introduzidas a seguir.

3.3.1. Amostragem simples

A amostragem simples consiste basicamente em extrair objetos, com ou sem reposição, do conjunto de dados original com igual probabilidade. No primeiro caso, quando há a reposição, um objeto pode ocorrer mais de uma vez em uma amostra e a probabilidade de ocorrência de cada objeto mantém-se constante durante todo o processo. No segundo

caso, quando não existe reposição, cada objeto poderá acontecer apenas uma vez em uma amostra. Assim, a probabilidade de ocorrência de cada objeto é alterada a cada passo do processo. O código abaixo apresenta um exemplo para cada caso.

```
import pandas

dados = pandas.read_csv('breast_cancer.csv')

# gerando uma amostra aleatoria simples de 100 elementos
# da base breast cancer, sem reposicao
dados.sample(100)

# gerando uma amostra aleatoria simples de 100 elementos
# da base breast cancer, com reposicao
dados.sample(100, replace=True)
```

3.3.2. Amostragem estratificada

A amostragem estratificada é tipicamente utilizada quando há o desbalanceamento na quantidade de exemplos de cada classe em um conjunto de dados. Ou seja, neste cenário, normalmente uma ou outra classe possuirá uma quantidade de objetos consideravelmente maior do que as demais. A fim de gerar um classificador que não seja tendencioso para uma ou mais classes majoritárias ou, de suavizar a dificuldade de modelar alguma das classes de um problema, a amostragem estratificada pode ser utilizada de duas maneiras.

Na primeira abordagem para amostragem estratificada, o subconjunto a ser extraído contém a mesma quantidade de objetos para cada classe. O código a seguir ilustra, de maneira simplificada, esse tratamento.

```
import pandas

dados = pandas.read_csv('breast_cancer.csv')

# tamanho da amostra estratificada
tamanho_amostra = 100

# obtendo as classes da base de dados
classes = dados['diagnosis'].unique()

# calculando a quantidade de amostras por classe
# neste exemplo, serao amostradas as mesmas
# quantidades para cada classe
qtde_por_classe = round(tamanho_amostra / len(classes))

# nesta lista armazenaremos, para cada classe, um
# pandas.DataFrame com suas amostras
amostras_por_classe = []
```

```
for c in classes:
    # obtendo os indices do DataFrame
    # cujas instancias pertencem a classe c
    indices_c = dados['diagnosis'] == c

    # extraindo do DataFrame original as observacoes da
    # classe c (obs_c sera um DataFrame tambem)
    obs_c = dados[indices_c]

    # extraindo a amostra da classe c
    # caso deseje-se realizar amostragem com reposicao
    # ou caso len(obs_c) < qtde_por_classe, pode-se
    # informar o parametro replace=True
    amostra_c = obs_c.sample(qtde_por_classe)

    # armazenando a amostra_c na lista de amostras
    amostras_por_classe.append(amostra_c)

# concatenando as amostras de cada classe em
# um único DataFrame
amostra_estratificada = pd.concat(amostras_por_classe)
```

Na segunda abordagem, a amostra gerada do conjunto de dados original mantém as mesmas proporções de objetos da base de dados original em cada classe. O exemplo de código a seguir demonstra como isso pode ser feito.

```
import pandas

dados = pandas.read_csv('breast_cancer.csv')

# tamanho da amostra estratificada
tamanho_amostra = 100

# obtendo as classes da base de dados
classes = dados['diagnosis'].unique()

# nesta lista armazenaremos, para cada classe, um
# pandas.DataFrame com suas amostras
amostras_por_classe = []

for c in classes:
    # obtendo os indices do DataFrame
    # cujas instancias pertencem a classe c
    indices_c = dados['diagnosis'] == c
```

```
        # extraindo do DataFrame original as observacoes da
        # classe c (obs_c sera um DataFrame tambem)
        obs_c = dados[indices_c]

        # calculando a proporcao de elementos da classe c
        # no DataFrame original
        proporcao_c = len(obs_c) / len(dados)

        # calculando a quantidade de elementos da classe
        # c que estarao contidos na amostra estratificada
        qtde_c = round(proporcao_c * tamanho_amostra)

        # extraindo a amostra da classe c
        # caso deseje-se realizar amostra com reposicao ou,
        # caso len(obs_c) < qtde_c, pode-se
        # informar o parametro replace=True
        amostra_c = obs_c.sample(qtde_c)

        # armazenando a amostra_c na lista de amostras
        amostras_por_classe.append(amostra_c)

# concatenando as amostras de cada classe em
# um único DataFrame
amostra_estratificada = pd.concat(amostras_por_classe)
```

3.4. Dados ausentes

Dados ausentes podem ocorrer por uma série de motivos em cenários de aplicação reais (veja, por exemplo, a discussão no Capítulo 3 do livro de [Faceli et al. 2011]). Uma alternativa que pode ser utilizada para a solução de tal problema é a remoção de objetos que contenham valores ausentes. Entretanto, deve-se perceber que uma grande quantidade de informações relevantes podem acabar sendo descartadas (por exemplo, se muitos objetos possuírem valores ausentes, tem-se uma redução significativa da base de dados). Portanto, uma alternativa bastante comum é a inserção automática de tais valores, utilizando alguma medida capaz de estimá-los. Comumente, tal medida consiste na média, mediana ou moda do respectivo atributo. Em Python, a inserção automática pode ser realizada por meio do código abaixo.

```
import pandas

# carrega uma versao do conjunto de dados contendo valores
# ausentes
dados = pandas.read_csv('breast_cancer_missing.csv')

# cada valor ausente eh indicado por um
# 'Not a Number' (numpy.nan)
```

```
# isso pode ser verificado conforme abaixo
# onde 'valores_ausentes' sera um DataFrame em que
# cada posicao indica se o valor esta ausente
# (True) ou nao (False)
valores_ausentes = dados.isnull()

# calculando as medias dos atributos
# 'medias' resulta em uma variavel do tipo 'Series',
# o qual consiste em uma implementacao da Pandas
# de um numpy.array com cada elemento contendo
# um rotulo (ou nome)
medias = dados.mean()

# por fim, a insercao de valores sera realizada pela
# funcao fillna, onde o parametro inplace=True indica
# para os valores serem inseridos no próprio DataFrame
# (caso seja falso, uma copia do DataFrame original sera
# retornada contendo os valores preenchidos)
dados.fillna(medias, inplace=True)
```

3.5. Dados redundantes

Dados redundantes podem ocorrer tanto para os objetos quanto para os atributos de um conjunto de dados [Faceli et al. 2011]. Quando há a redundância de objetos, dois ou mais deles possuem valores muito similares (ou até mesmo iguais) para todos os seus atributos. Isso pode ser um problema pois, ao aplicar um algoritmo de AM, tais objetos irão inserir uma ponderação artificial aos dados. Objetos redundantes podem ser removidos por meio do código a seguir.

```
import pandas

# carregando uma versao do conjunto de dados contendo
# objetos redundantes (repetidos)
dados = pandas.read_csv('breast_cancer_red.csv')

# removendo exemplos redundantes
# observe que o resultado sera a base original
# o parametro inplace determina se a alteracao deve
# ocorrer no proprio DataFrame
# portanto, caso inplace=False, a funcao drop_duplicates
# retorna uma copia de breast_cancer_red com os exemplos
# redundantes removidos
dados.drop_duplicates(inplace=True)
```

Para o caso de redundância de atributos, tem-se que um deles pode estar fortemente correlacionado com outro, o que indica um comportamento similar de suas variações. Algumas vezes, em tais casos, o valor de um atributo pode ser calculado ou obtido

por meio de um ou mais dentre os atributos remanescentes. A remoção de atributos redundantes é proposta como exercício na Seção 3.9. Portanto, o respectivo exemplo prático foi omitido neste texto.

3.6. Ruídos

Ruídos normalmente consistem em valores que aparentemente não pertencem à distribuição que gerou o conjunto de dados à disposição [Faceli et al. 2011]. Vários podem ser os motivos para a ocorrência dos mesmos, desde erros na digitação ao tabular os dados, problemas de medição em instrumentos utilizados para coletar os dados ou, até mesmo, valores pouco comuns mas reais. Portanto, ao descartar objetos que apresentem valores ruidosos para um ou mais atributos, pode-se perder informações relevantes para o problema estudado.

A fim de reduzir a influência de ruídos nos atributos do conjunto de dados estudado, diversas técnicas podem ser aplicadas. Dentre elas, uma das mais simples consiste em dividir os valores de cada atributo em faixas, de modo que cada faixa contenha aproximadamente a mesma quantidade de valores. Em seguida, os valores contidos em cada faixa são substituídos por alguma medida que os sumarize como, por exemplo, a média. Um exemplo desta técnica é demonstrado a seguir.

```
import pandas

dados = pandas.read_csv('breast_cancer.csv')

# dividindo o atributo 'mean_radius' em 10 faixas
bins = pandas.qcut(dados['mean_radius'], 10)

# a quantidade de valores aproximadamente igual em
# cada faixa pode ser comprovada pelo metodo
# value_counts
bins.value_counts()

# O metodo groupby permite que valores contidos na
# coluna de um DataFrame sejam agrupados segundo
# algum criterio.
# Neste exemplo, a coluna 'mean_radius' sera agrupada
# pelas faixas definidas pelo metodo qcut.
grupos = dados['mean_radius'].groupby(bins)

# obtendo a media de cada faixa
medias = grupos.mean()

# Obtendo a nova coluna.
# O metodo apply recebe como parametro uma funcao,
# aplica tal funcao a todos os seus elements e retorna
# um pandas.Series contendo os resultados.
```

```
# Neste caso, cada elemento de bins consiste
# no intervalo que o respectivo valor de 'mean_radius'
# pertence e, assim, a funcao informada em apply
# retornara a respectiva media de cada intervalo.
novo_mean_radius = bins.apply(lambda x : medias[x])

# por fim, a coluna 'mean_radius' do DataFrame original
# eh atualizada
dados['mean_radius'] = novo_mean_radius
```

3.7. Transformação de dados

3.7.1. Transformação de dados simbólicos para dados numéricos

Diversas técnicas de AM exigem que os dados de entrada consistam apenas em valores numéricos. Entretanto, diversos conjuntos reais podem apresentar atributos qualitativos nominais ou ordinais, tornando assim necessário o emprego de técnicas que permitam a conversão de tais atributos.

No primeiro caso, quando houver a existência de atributos nominais, a inexistência de qualquer ordem deve persistir na conversão do atributo. Uma abordagem bastante conhecida e utililizada para isso consiste na codificação 1-de-c [Faceli et al. 2011]. Nela, considerando que um atributo possua c valores possíveis, são criados c novos atributos binários, onde cada posição indica um possível valor do atributo nominal original. Desse modo, apenas uma posição da nova sequência binária de cada objeto poderá ser igual a 1, indicando qual é o valor correspondente de um determinado objeto para o atributo original. Em Python, tal conversão pode ser facilmente realizada por meio do método `pandas.get_dummies`, conforme demonstrado pelo exemplo a seguir.

```
import pandas

dados = pandas.read_csv('cmc.csv')

# O atributo husband_occupation consiste em um atributo
# nominal com 4 valores possiveis.
# A conversao do mesmo para a codificacao 1-de-c eh
# feita como:
occ1dec = pandas.get_dummies(dados['husband_occupation'])
```

Quando houverem atributos qualitativos ordinais, pode-se também optar por representações binárias. Entretanto, as mesmas deverão ser diferentes da representação 1-de-c, uma vez que as distâncias entre os possíveis valores de um atributo não serão mais iguais para todos os casos. Para isso, pode-se utilizar o código cinza ou o código termômetro [Faceli et al. 2011].

O código cinza consiste na transformação dos valores inteiros para as suas respectivas representações binárias. Em Python, tal transformação pode ser realizada conforme segue.

```
import pandas

dados = pandas.read_csv('cmc.csv')

# O atributo wife_education consiste em um atributo
# qualitativo ordinal com 4 valores possiveis.
# A conversao do mesmo para o codigo cinza pode ser
# feita pela aplicacao do metodo 'bin' na respectiva
# coluna do DataFrame. O metodo 'bin' recebe como entrada
# um valor inteiro e retorna uma string com a representacao
# binaria do mesmo.
wife_ed_binario = dados['wife_education'].apply(bin)
```

O código termômetro realiza a transformação do respectivo atributo qualitativo ordinal em um vetor de c posições, onde c indica a quantidade de valores possíveis. Desse modo, cada valor ordinal corresponde a um vetor binário preenchido com uma quantidade de valores iguais a 1, acumulados sequencialmente da esquerda para a direita ou vice-versa, equivalente à sua posição na ordem dos valores possíveis do atributo original. Esta transformação é proposta como exercício na Seção 3.9. Portanto, o respectivo código foi omitido.

3.7.2. Transformação de dados numéricos para dados simbólicos

Várias de técnicas de AM como, por exemplo, alguns algoritmos de árvore de decisão, exigem que os dados de entrada assumam valores qualitativos. Portanto, em cenários nos quais há a presença de atributos com valores contínuos, torna-se importante a aplicação de técnicas adequadas de discretização.

Existe uma grande quantidade de técnicas de discretização disponíveis na literatura (os estudos de [Kotsiantis e Kanellopoulos 2006, Garcia et al. 2013] sumarizam várias delas), as quais são baseadas em diferentes suposições e procedimentos. Dentre elas, as mais simples e intuitivas consistem na divisão de um intervalo original de valores por faixas, podendo estas terem a mesma largura ou frequências de valores similares. O código a seguir ilustra ambas.

```
import pandas

dados = pandas.read_csv('breast_cancer.csv')

# Obtendo a coluna 'mean_radius'.
mean_radius = dados['mean_radius']

# Discretizando a coluna 'mean_radius'.
# O parametro 'bins' define em quantos intervalos
# sera discretizado.
# O parametro labels define o rotulo de cada
# intervalo.
```

```
# Neste caso, como labels eh igual a range(10),
# os intervalos serao discretizados com valores
# inteiros entre 0 e 9.
# O metodo pandas.cut discretiza em intervalos
# de larguras iguais. Caso deseje-se discretizar
# com frequencias iguais, deve-se utilizar o
# metodo pandas.qcut.
mean_radius_disc = pandas.cut(dados, bins=10,
                              labels=range(10))

# Atualizando a respectiva coluna no DataFrame
# original.
dados['mean_radius'] = mean_radius_disc
```

3.7.3. Normalização de atributos numéricos

Muitos conjuntos de dados reais apresentam atributos contínuos cujos valores espalham-se por distintas faixas de valores ou que possuem diferentes variações de valores, devido às suas naturezas ou escalas em que foram medidas. Em alguns problemas, tais diferenças podem ser importantes e devem ser levadas em conta [Faceli et al. 2011]. Entretanto, em outras situações pode ser necessária uma normalização dos valores de cada atributo, a fim de que se evite que algum predomine sobre outro ou que inclua qualquer tipo de ponderação indesejada ao induzir um modelo de AM. Os tipos mais comuns de normalização consistem em reescala ou padronização.

A normalização por reescala define, através de um valor mínimo e um valor máximo, um novo intervalo onde os valores de um atributo estarão contidos. Tipicamente, tal intervalo é definido como $[0,1]$. Portanto, para este caso, a normalização por reescala de um atributo j de um objeto x_i pode ser calculada como:

$$x_{ij} = \frac{x_{ij} - \min_j}{\max_j - \min_j} \tag{1}$$

sendo $\min_j$ e $\max_j$, nessa ordem, os valores mínimo e máximo do atributo j para o conjunto de dados considerado.

Na normalização por padronização, os diferentes atributos contínuos poderão abranger diferentes intervalos, mas deverão possuir os mesmos valores para alguma medida de posição e de espalhamento/variação [Faceli et al. 2011]. Tipicamente, tais medidas irão consistir na média e no desvio-padrão. Neste caso, o valor normalizado de um atributo j em um objeto i é dado por:

$$x_{ij} = \frac{x_{ij} - \bar{x}_{\cdot j}}{s_j} \tag{2}$$

onde $\bar{x}_{\cdot j}$ e s_j representam a média do atributo j e o seu desvio-padrão, respectivamente. Desse modo, cada cada atributo j terá média zero e desvio-padrão unitário.

Os dois tipos de normalização supramencionados podem ser executados conforme o código em seguida.

```
import pandas
from sklearn.preprocessing import minmax_scale
from sklearn.preprocessing import scale

dados = pandas.read_csv('breast_cancer.csv')

# Obtendo os nomes das colunas do DataFrame
# como uma lista.
cols = list(dados.columns)

# Removendo da lista 'cols' os nomes
# 'sample_id' e 'diagnosis' que sao
# colunas que nao serao normalizadas
cols.remove('sample_id')
cols.remove('diagnosis')

# Copiando os dados e aplicando a normalizacao
# por reescala nas colunas do DataFrame que contem
# valores continuos.
# Por padrao, o metodo minmax_scale reescala
# com min=0 e max=1.
dados_amp = dados.copy()
dados_amp[cols] = dados[cols].apply(minmax_scale)

# Copiando os dados e aplicando a normalizacao
# por padronização a todas as colunas do DataFrame.
# Por padrao, o metodo scale subtrai a media e
# divide pelo desvio-padrao.
dados_dist = dados.copy()
dados_dist[cols] = dados[cols].apply(scale)
```

3.8. Redução de dimensionalidade

Muitos dos conjuntos de dados tratados em mineração de dados possuem como característica um elevado número de atributos. Uma aplicação biológica clássica em que esse tipo de situação ocorre é na análise de dados de expressão gênica, onde milhares de genes são tipicamente aferidos em um número consideravelmente menor e mais limitado de amostras (normalmente de dezenas a algumas centenas).

São poucas as técnicas de AM que são capazes de lidar com uma grande quantidade de atributos de maneira eficiente (tanto do ponto de vista computacional quanto do ponto de vista de acurácia). Portanto, a utilização de técnicas que sejam capazes de reduzir a dimensionalidade dos dados, por meio de agregação ou seleção de atributos, pode auxiliar na indução de um modelo de AM. Nesta seção, serão abordados dois exemplos para redução de dimensionalidade: *Principal Component Analysis*, para agregação de atributos; e filtragem por um limiar de variância pré-estabelecido, para seleção de atributos.

3.8.1. *Principal Component Analysis* (PCA)

O PCA consiste em uma técnica estatística que utiliza uma transformação linear para reexpressar um conjunto de atributos em um conjunto menor de atributos não correlacionados linearmente, mantendo boa parte das informações contidas nos dados originais [Dunteman 1989]. Em suma os objetivos do PCA consistem em [Abdi e Williams 2010]:

- Extrair apenas as informações mais relevantes de um conjunto de dados;
- Comprimir o tamanho do conjunto de dados original, simplificando assim sua descrição; e
- Permitir a análise da estrutura dos objetos e dos atributos de um conjunto de dados.

Um simples e intuitivo tutorial sobre o PCA está disponível em `https://arxiv.org/abs/1404.1100`. Abaixo é apresentado um exemplo da aplicação do PCA no conjunto *Breast Cancer*.

```
import pandas
from sklearn.decomposition import PCA

dados = pd.read_csv('breast_cancer.csv')

# Obtendo os nomes das colunas.
cols = list(dados.columns)

# Removendo colunas que nao serao inclusas
# na reducao de dimensionalidade.
cols.remove('sample_id')
cols.remove('diagnosis')

# Instanciando um PCA. O parametro n_components
# indica a quantidade de dimensoes que a base
# original sera reduzida.
pca = PCA(n_components=2, whiten=True)

# Aplicando o pca na base breast_cancer.
# O atributo 'values' retorna um numpy.array
# de duas dimensões (matriz) contendo apenas
# os valores numericos do DataFrame.
dados_pca = pca.fit_transform(dados[cols].values)

# O metodo fit_transform retorna outro numpy.array
# de dimensao numero_objetos x n_components.
# Apos isso, instancia-se um novo DataFrame contendo
# a base de dados original com dimensionalidade
# reduzida.
```

```
dados_pca = pd.DataFrame(dados_pca,
                         columns=['comp1', 'comp2'])
```

3.8.2. Filtragem por limiar de variância

A filtragem por limiar de variância consiste basicamente em manter apenas os atributos da base de dados original que possuam variância acima de um valor pré-estabelecido. O código a seguir ilustra o funcionamento da mesma em Python.

```
import pandas

dados = pd.read_csv('breast_cancer.csv')

# Obtendo os nomes das colunas.
cols = list(dados.columns)

# Removendo colunas que nao serao inclusas
# na reducao de dimensionalidade.
cols.remove('sample_id')
cols.remove('diagnosis')

# Instanciando um VarianceThreshold. Esta
# classe recebe apenas um parâmetro real,
# que indica o valor de limiar desejado.
var_thr = VarianceThreshold(1.0)

# Selecionando apenas os atributos com
# variância maior ou igual a 1.
# 'dados_var_thr' sera um numpy.array com
# duas dimensoes.
dados_var_thr = var_thr.fit_transform(dados[cols].values)
```

Outros métodos mais sofisticados para seleção de atributos estão disponíveis na biblioteca scikit-learn por meio do módulo `sklearn.feature_selection`[3].

3.9. Exercícios

1. Observe que os códigos apresentados na Seção 3.3.2 nem sempre retornam uma amostra com o tamanho desejado exato (dica: teste ambos os códigos com a base Iris, por exemplo). Por que isso acontece? Escreva uma função que receba como entrada um `DataFrame`, um tamanho de amostra desejado, o tipo de amostragem estratificada a ser feita e retorne uma amostra estratificada com o tamanho exato desejado. Quais as implicações desta nova função no resultado da amostragem?

2. Repare que o código apresentado na Seção 3.4 considera os exemplos de todas as classes para o cálculo dos valores a serem inseridos no `DataFrame`. Escreva

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

uma função que calcule e insira tais valores separadamente para cada classe. Ademais, tal função deverá receber como parâmetro a medida que será calculada (média ou mediana). Considerando a base de dados `breast_cancer_missing.csv`, crie um arquivo CSV com o novo `DataFrame` gerado para cada caso.

3. Escreva uma função que receba como entrada um `DataFrame`, calcule a correlação entre todos os seus atributos (através da função `DataFrame.corr`) e retorne um novo `DataFrame` mantendo apenas um atributo dentre aqueles que possuem correlação acima (ou abaixo, caso a correlação seja negativa) de um limiar informado como parâmetro.

4. Implemente a representação pelo código termômetro descrita na Seção 3.7.1 e aplique a mesma para todos os atributos qualitativos ordinais do conjunto de dados *Contraceptive Method Choice*.

5. Repare que no código apresentado na Seção 3.8 o novo `DataFrame` gerado possui duas colunas nomeadas `'comp1'` e `'comp2'`. Pesquise sobre o PCA e descreva com as suas palavras o que esses dois novos atributos representam.

6. Aplique o PCA no conjunto de dados Iris, introduzido no capítulo anterior, com `n_components=2`. Gere um *scatter plot* com diferentes cores para cada classe. O que pode ser observado? Como o PCA pode auxiliar como ferramenta de visualização?

Referências

[Abdi e Williams 2010] Abdi, H. e Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459.

[Dunteman 1989] Dunteman, G. H. (1989). *Principal components analysis*. Number 69. Sage.

[Faceli et al. 2011] Faceli, K., Lorena, A. C., Gama, J., e Carvalho, A. C. P. L. F. (2011). Inteligência artificial: Uma abordagem de aprendizado de máquina. *Rio de Janeiro: LTC*, 2:192.

[Garcia et al. 2013] Garcia, S., Luengo, J., Sáez, J. A., Lopez, V., e Herrera, F. (2013). A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750.

[Kotsiantis e Kanellopoulos 2006] Kotsiantis, S. e Kanellopoulos, D. (2006). Discretization techniques: A recent survey. *GESTS International Transactions on Computer Science and Engineering*, 32(1):47–58.

[Lim et al. 2000] Lim, T.-S., Loh, W.-Y., e Shih, Y.-S. (2000). A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine learning*, 40(3):203–228.

[Mangasarian et al. 1995] Mangasarian, O. L., Street, W. N., e Wolberg, W. H. (1995). Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577.

[Street et al. 1992] Street, W. N., Wolberg, W. H., e Mangasarian, O. L. (1992). Nuclear feature extraction for breast tumor diagnosis.