Técnicas Clássicas de Reconhecimento de Padrões (2020/01)

Exercício 02 - Classificação da Base de Dados de Leucemia

Reprodução dos Resultados Apresentados por Golub et al 1999

Aluno: Ramon Gomes Durães de Oliveira (2019720188)

Pré-processamento de dados

Carregando os Dados

Foram utilizados os dados disponibilizados pela plataforma Kaggle.

- "class_df": contém as classes de cada observação
- "train_df": contém as observações do conjunto de treinamento
- "test_df": contém as observações do conjunto de teste

In [247]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

class_df = pd.read_csv('./data/actual.csv')
train_df = pd.read_csv('./data/data_set_ALL_AML_train.csv')
test_df = pd.read_csv('./data/data_set_ALL_AML_independent.csv')
```

Limpando os Dados

Nesta seção os dados são limpados, removendo colunas desnecessárias e transpondo os dados. Além disso, a classe de cada observação é extraída do conjunto de dados "class_df" e integrada aos dados de treinamento e teste. As classes do conjunto de teste serão utilizadas apenas para avaliação da classificação. Os conjuntos de dados resultantes possuem 38 observações de treinamento e 34 de teste, como no artigo do Golub. Há dados para 7129 genes.

In [248]:
```
valid_columns = [col for col in train_df.columns if "call" not in col]
train_df = train_df[valid_columns]
train_df = train_df.T
train_df = train_df.drop(['Gene Description','Gene Accession Number'],axis=0)
train_df.index = pd.to_numeric(train_df.index)
train_df.sort_index(inplace=True)
class_dict = {'AML':0,'ALL':1}
train_df['class'] = class_df[:38]['cancer'].replace(class_dict).values
train_df.head()
```

Out[248]:

	0	1	2	3	4	5	6	7	8	9	...	7120	7121	7122	7123	7124
1	-214	-153	-58	88	-295	-558	199	-176	252	206	...	511	-125	389	-37	793
2	-139	-73	-1	283	-264	-400	-330	-168	101	74	...	837	-36	442	-17	782
3	-76	-49	-307	309	-376	-650	33	-367	206	-215	...	1199	33	168	52	1138
4	-135	-114	265	12	-419	-585	158	-253	49	31	...	835	218	174	-110	627
5	-106	-125	-76	168	-230	-284	4	-122	70	252	...	649	57	504	-26	250

5 rows × 7130 columns

Dados de teste:

In [249]:
```
valid_columns = [col for col in test_df.columns if "call" not in col]
test_df = test_df[valid_columns]
test_df = test_df.T
test_df = test_df.drop(['Gene Description','Gene Accession Number'],axis=0)
test_df.index = pd.to_numeric(test_df.index)
test_df.sort_index(inplace=True)
test_df['class'] = class_df[38:]['cancer'].replace(class_dict).values
test_df.head()
```

Out[249]:

	0	1	2	3	4	5	6	7	8	9	...	7120	7121	7122	7123	7124
39	-342	-200	41	328	-224	-427	-656	-292	137	-144	...	1023	67	214	-135	1074
40	-87	-248	262	295	-226	-493	367	-452	194	162	...	529	-295	352	-67	67
41	-62	-23	-7	142	-233	-284	-167	-97	-12	-70	...	383	46	104	15	245
42	22	-153	17	276	-211	-250	55	-141	0	500	...	399	16	558	24	893
43	86	-36	-141	252	-201	-384	-420	-197	-60	-468	...	91	-84	615	-52	1235

5 rows × 7130 columns

Seleção e Classificação

Seleção de Genes Relevantes

Para selecionar um subconjunto de genes relevantes e efetivos para a classificação, no artigo os autores criam dois genes simbólicos com o grau de expressão máximo para uma das classes e mínimo para a outra.

Nos dados utilizados pelo autor, os valores máximos e mínimos correspondiam ao range de 0 a 1. Neste conjunto de dados este não é o caso. Logo, será utilizado o máximo e mínimo amostral do nível de expressão de cada gene.

No artigo, a correlação de um gene com a classificação desejada é calculada utilizando a fórmula:

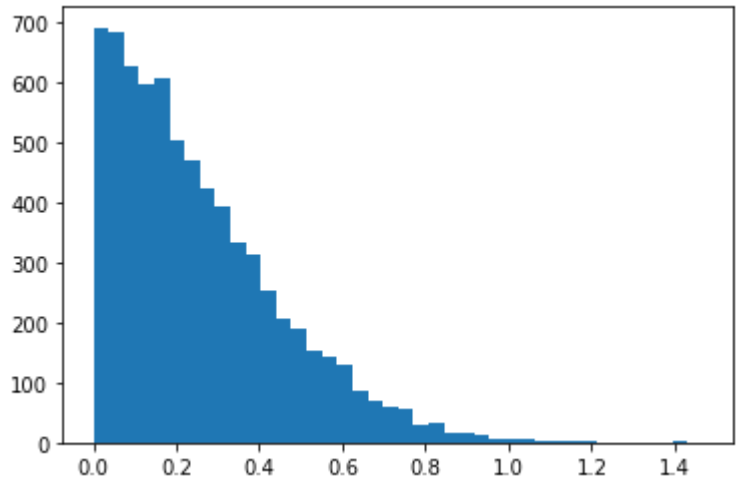
$$P(g,c) = [\mu_1(g) - \mu_2(g)]/[\sigma_1(g) + \sigma_2(g)]$$

Na qual:

- g é um gene
- μ é a média amostral
- σ é o desvio padrão amostral
- os números subscritos (μ_1,μ_2) indicam as classes 1 e 2.

Aqui, busca-se por valores altos de $|P(g,c)|$. Essa medida será calculada para todos os genes e eles serão ordenados de acordo com seu valor absoluto:

In [250]:
```
mu1 = train_df[train_df['class']==0].iloc[:,:-1].mean()
sigma1 = train_df[train_df['class']==0].iloc[:,:-1].std()
mu2 = train_df[train_df['class']==1].iloc[:,:-1].mean()
sigma2 = train_df[train_df['class']==1].iloc[:,:-1].std()
Pgc = (mu1 - mu2) / (sigma1 + sigma2)
abs_Pgc = np.abs(Pgc)
plt.hist(abs_Pgc, bins = 40);
```



O artigo diz que o número de genes selecionado foi de certa forma arbitrário. Eles mantiveram 50 para ter um classificador razoavelmente robusto mas a performance foi semelhante para diversos subconjuntos de genes. Selecionando os 50 genes com maior valor absoluto de $P(g, c)$, estabelece-se um threshold em 0.91. Os genes selecionados foram:

In [251]:
```
selected_genes = abs_Pgc>.91
selected_genes = selected_genes.index[selected_genes.values]
selected_genes
```

Out[251]:
```
Index([ 148,  460, 1248, 1305, 1629, 1673, 1744, 1806, 1833, 1881, 2019, 204
2,
       2110, 2120, 2185, 2241, 2266, 2287, 2347, 2353, 2401, 2440, 2641, 275
8,
       3055, 3257, 3319, 3846, 4051, 4176, 4195, 4229, 4327, 4534, 4846, 503
8,
       5500, 5592, 5771, 6199, 6200, 6280, 6372, 6375, 6470, 6538, 6802, 680
5,
       6854, 6973],
      dtype='object')
```

Nota-se que agora o conjunto de dados tem 38 observações mas apenas 50 genes.

In [252]:
```
train_df[selected_genes].shape
```

Out[252]:
```
(38, 50)
```

Classificação por Votação

No artigo, a classificação é feita utilizando apenas o conjunto de dados de treinamento e o subconjunto selecionado de genes. Para cada gene deste subconjunto, parâmetros (a_g, b_g) são definidos de forma que:

- $a_g = P(g, c)$: reflete a correlação enter os níveis de expressão do gene g e a distinção de classes;
- $b_g = [\mu_1(g) + \mu_2(g)]/2$: é a média do nível de expressão média entre as classes.

Considerando uma nova observação X a ser classificada, x_g é o nível de expressão do gene g na observação. Dessa forma, o voto do gene g é definido por:

$$v_g = a_g(x_g - b_g)$$

Com um valor positivo indicando um voto para a classe 1 e um valor negativo indicando um voto para a classe 2. Os totais de votos para cada classe são obtidos somando os votos individuais.

Erro de Validação Cruzada (Leave-One-Out):

Para validar os resultados obtidos para o classficador durante o treinamento, os autores utilizam validação cruzada do estilo "leave-one-out": para cada observação, treina-se o classificador nas $n-1$ observações restantes e avalia-se o resultado na observação selecionada. O total de erros é considerado o erro de validação cruzada. Desta forma, utiliza-se todo o conjunto de treinamento para treinar e para testar a performance do classificador.

Abaixo, o processo de seleção de genes será repetido e o erro de validação será obtido utilizando esta técnica:

In [330]:
```
n = train_df.shape[0]
valid_pred = []

for i in np.arange(1, n+1):
    valid_train_df = train_df[~train_df.index.isin([i])]
    valid_test_df = train_df[train_df.index.isin([i])]

    mu1 = valid_train_df[valid_train_df['class']==0].iloc[:,:-1].mean()
    sigma1 = valid_train_df[valid_train_df['class']==0].iloc[:,:-1].std()
    mu2 = valid_train_df[valid_train_df['class']==1].iloc[:,:-1].mean()
    sigma2 = valid_train_df[valid_train_df['class']==1].iloc[:,:-1].std()
    Pgc = (mu1 - mu2) / (sigma1 + sigma2)
    abs_Pgc = np.abs(Pgc)

    ag = Pgc[selected_genes]
    bg = (mu1[selected_genes] + mu2[selected_genes])/2
    V = ag * (valid_test_df[selected_genes] - bg) # matriz com os votos de cad
a gene para cada observação
    V1 = np.sum(V>0, axis=1) # soma dos votos para uma classe
    V2 = V.shape[1] - V1 # soma dos votos para a outra classe

    pred = 1 * (V1 <= V2)

    valid_pred.append(pred.values[0])

confusion_matrix(train_df.iloc[:,-1], valid_pred)
```

Out[330]:
```
array([[11,  0],
       [ 0, 27]], dtype=int64)
```

A matriz de confusão acima mostra que todas as observações foram classificadas corretamente no conjunto de validação!

Erro de teste:

Agora, o classificador treinado no conjunto de treinamento é avaliado para o conjunto de teste:

In [254]:
```
ag = Pgc[selected_genes]
bg = (mu1[selected_genes] + mu2[selected_genes])/2
V = ag * (test_df[selected_genes] - bg) # matriz com os votos de cada gene par
a cada observação
V1 = np.sum(V>0, axis=1) # soma dos votos para uma classe
V2 = 50 - V1 # soma dos votos para a outra classe
pred = 1 * (V1 <= V2)

confusion_matrix(test_df.iloc[:,-1], pred)
```

Out[254]:
```
array([[11,  3],
       [ 0, 20]], dtype=int64)
```

A matriz de confusão acima mostra que, das 34 observações do conjunto de teste, 31 foram classificadas corretamente utilizando a abordagem do artigo. Para a classe 0 (AML), 03 observações foram classificadas erroneamente, enquanto para a classe 1 (ALL) todas foram corretamente classificadas.

A matriz de confusão acima denota o erro de validação cruzada obtido através da técnica Leave-One-Out. Desta vez, das 38 observações do conjunto de treinamento.

Prediction Strength (PS)

Para aumentar a certeza da classificação, o artigo define um índice chamado Prediction Strength ("Força da Predição", PS) de acordo com a fórmula abaixo:

- $PS = (V_{win} - V_{lose})/(V_{win} + V_{lose})$

em que V_{win} é o total de votos da cclasse vencedora e V_{lose} é o total de votos da classe perdedora. A medida reflete a margem de vitória. Caso ela seja menor que 0.3, a classe da observação é dada como indefinida.

Abaixo é calculada a métrica PS para todas as observações.

In [288]:
```
Vwin = pd.concat([V2[pred==1], V1[pred==0]]).sort_index()
Vlose = pd.concat([V2[pred==0], V1[pred==1]]).sort_index()
PS = (Vwin - Vlose) / (Vwin + Vlose)
sum(PS > 0.3)
```

Out[288]:
```
26
```

A escolha do threshold de PS como 0.3 feita no artigo faz com que, das 34 observações de treinamento, 26 tenham PS significativo. A escolha do threshold de PS como 0.3 é baseada numa análise anterior que era hospedada no site da organização do autor mas que provavelmente já foi retirado de lá nos últimos 20 anos (não encontrei).

Calculando a proporção de acertos para as observações com PS > 0.3:

In [292]:
```
confusion_matrix(test_df[PS > 0.3].iloc[:,-1], pred[PS > 0.3])
```

Out[292]:
```
array([[ 7,  0],
       [ 0, 19]], dtype=int64)
```

Para as predições com índice de força PS acima de 0.3, todas as observações são classificadas corretamente!

Extra: Classificação por SOMs

Após a classificação por votação majoritária, o artigo cita ainda uma classificação utilizando Self-Organizing Maps (SOMs). Para fazer uma clusterização similar à que foi feita no artigo, será utilizado um KNN simples com 2 clusters. A performance primeiro será avaliada no conjunto de treinamento e em seguida no conjunto de teste.

In [273]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

knn = KNeighborsClassifier(n_neighbors=2, metric='euclidean')
knn.fit(train_df.iloc[:,:-1], train_df.iloc[:,-1])
train_pred = knn.predict(train_df.iloc[:,:-1])
confusion_matrix(train_df.iloc[:,-1], train_pred)
```

Out[273]:
```
array([[11,  0],
       [ 0, 27]], dtype=int64)
```

A matriz de confusão do conjunto de treinamento acima mostra que todas as observações foram classificadas corretamente no conjunto de treinamento.

Avaliemos agora a performance no conjunto de teste:

In [274]:
```
test_pred = knn.predict(test_df.iloc[:,:-1])
confusion_matrix(test_df.iloc[:,-1], test_pred)
```

Out[274]:
```
array([[14,  0],
       [ 4, 16]], dtype=int64)
```

Para o conjunto de teste, a classificação acertou 30 das 34 observações.