COMPARANDO EFICIÊNCIA DE ALGORITMOS DE CLASSIFICAÇÃO PARA UMA DETERMINADA BASE DE DADOS

Fábio Oliviera Tempesta

Instituto Federal de Minas Gerais (IFMG) – Campus Bambuí

fabio.oliveira.tempesta@gmail.com

1 INTRODUÇÃO

A classificação de dados é uma tecnologia que vem sendo usada cada vez mais nas empresas, auxiliando-as em tomadas de decisões, produtividade, detecção de problemas, etc. Os algoritmos de classificação são aprendizados de máquina supervisionado que tenta designar o grupo correto de um objeto ou uma classe predeterminada, baseando nos seus atributos (FISHER; HAPANYENGWI, 1993; ZHANG, 2000). Eles funcionam da maneira que, de acordo com Bonifácio (2010), utilizam técnicas para aprender a partir de um conjunto de treinamento formado por conjunto de registros, e esse aprendizado é testado no conjunto de treinamento, tentando prever a classe correta em que um dos os registros inéditos pertencem.

A preparação de dados é uma etapa da mineração de dados que consiste em manipular a base de dados, limpando e transformando-a em dados melhores de serem processados aos olhos dos classificadores. Nong (2003) fomenta a importância dessa preparação, e que nem sempre é tão claro efetuá-la, ela pode levar, pelo menos, 60% do tempo de qualquer projeto de mineração. Vale ressaltar que os classificadores podem apresentar necessidades diferentes, dependendo da base de dados que aquele algoritmo trabalha melhor.

O presente trabalho consiste em comparar eficiência de alguns algoritmos de classificação, seguindo as seguintes determinações:

- Serão avaliados três classificadores: *Perceptron* Multicamadas, Floresta Aleatória e Gradiente Descendente Estocástico;
- Todos os algoritmos processarão a mesma base de dados, terão que classificá-la na versão original e nas versões pré-processadas;
- O particionamento dos dados de treino e teste será feita usando o método validação cruzada com 10 partições;
- Todo código de desenvolvimento será desenvolvido em *Python*, utilizando as bibliotecas *Pandas* e *Scikit Learn*, e disponibilizado no Apêndice A;
- As medidas de desempenho dos classificadores consistirá na acurácia e tempo de treinamento, utilizando as mesmas partições da validação cruzada;

- Devido a possível variação de resultados, os classificadores passarão por 15 testes, o resultado será média deles.

2 ANÁLISE E PREPARAÇÃO DA BASE DE DADOS

2.1 Análise da base de dados

A base de dados escolhida para o trabalho contém 1000 instâncias e 9 atributos (contando com a classe). A Tabela 1 mostra algumas informações sobre ela.

Tabela 1 – Representação da base de dados.

	ID	A1	A2	A3	A4	A5	A6	A7	A8	CLASS
0	1	False	9.6	94	True	True	False	False	0.294057	c0
1	2	False	5.2	74	False	False	True	True	0.204914	c0
2	3	False	2.2	43	False	True	False	True	0.655174	c2
3	4	True	10.0	63	True	True	False	False	0.308093	c0
4	5	True	2.0	25	True	False	True	True	0.836315	c1
...	...	...	...	...	...	...	...	...	...	...
995	996	True	2.9	60	False	False	False	False	0.132663	c0
996	997	False	2.6	69	True	True	True	True	0.714479	c2
997	998	False	3.6	41	True	True	False	True	0.612487	c2
998	999	True	1.1	72	True	True	True	False	0.794731	c0
999	1000	False	1.8	76	False	True	True	True	0.103138	c2

Fonte: Elabora pelo autor.

Inicialmente já percebe-se um atributo irrelevante para classificação, o ID da instância não trará nenhuma informação útil para nenhum classificador. Esse atributo será removido e se tornará a nova base de dados original. A base de dados não possui nenhuma instância duplicada ou valores faltantes, como mostra a Figura 1.

Devido a quantidade relativamente pequena de instâncias da base de dados, não será utilizado técnicas de redução de registros.

2.2 Preparação 1

Fazendo alguns testes, foi observado os valores máximos e mínimos dos atributos contínuos, conforme a Tabela 2.

Tabela 2 – Valor máximo e mínimo dos atributos contínuos.

Atributo	Mínimo	Máximo
A2	0	10
A3	0	100
A8	0	1

Fonte: Elabora pelo autor.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A1      1000 non-null   bool
 1   A2      1000 non-null   float64
 2   A3      1000 non-null   int64
 3   A4      1000 non-null   bool
 4   A5      1000 non-null   bool
 5   A6      1000 non-null   bool
 6   A7      1000 non-null   bool
 7   A8      1000 non-null   float64
 8   CLASS   1000 non-null   object
dtypes: bool(5), float64(2), int64(1), object(1)
memory usage: 36.3+ KB
```

Figura 1 – Informações sobre a base de dados.
Fonte: Elaborada pelo autor.

Foi feita uma transformação de variáveis com o intuito de evitar discrepâncias entre os valores das variáveis contínuas. O atributo A2 será dividido por 10 e A3 por 100. A tabela 3 mostra esses dados após a transformação.

Tabela 3 – Base de dados com a preparação 1

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**	**CLASS**
0	False	0.96	0.94	True	True	False	False	0.294057	c0
1	False	0.52	0.74	False	False	True	True	0.204914	c0
2	False	0.22	0.43	False	True	False	True	0.655174	c2
3	True	1.00	0.63	True	True	False	False	0.308093	c0
4	True	0.20	0.25	True	False	True	True	0.836315	c1
..	...	...	...	...	...	...	...	...	...
995	True	0.29	0.60	False	False	False	False	0.132663	c0
996	False	0.26	0.69	True	True	True	True	0.714479	c2
997	False	0.36	0.41	True	True	False	True	0.612487	c2
998	True	0.11	0.72	True	True	True	False	0.794731	c0
999	False	0.18	0.76	False	True	True	True	0.103138	c2

Fonte: Elaborada pelo autor.

2.3 Preparação 2

Com o auxílio do software Weka (*Waikato Environment for Knowledge Analysis*), foi utilizado o filtro "MergeNominalValues"e obteve um bom desempenho dos classificadores removendo os atributos A4 e A6, e foi criado dois novos atributos com valores únicos (Figura 2), o que não ajudaria muito para classificação. Como ele teve um bom desempenho removendo esses atributos, a preparação 2 consistirá em uma base de dados sem eles, como mostra a tabela

4.

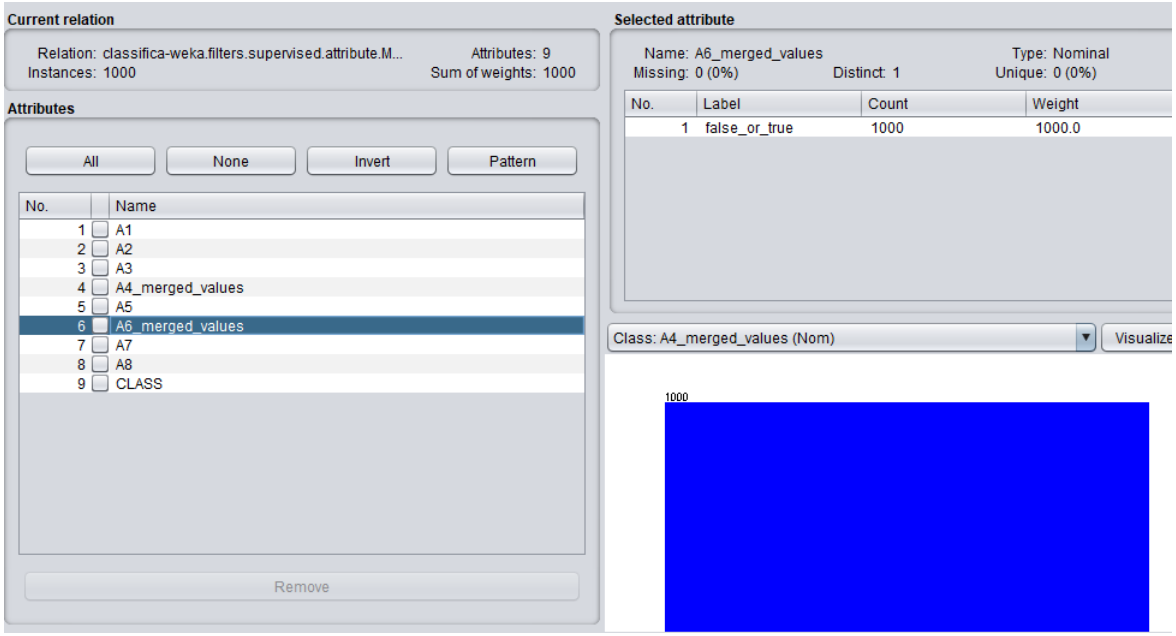

Figura 2 – Capura de tela do Weka.
Fonte: Elaborada pelo autor.

Tabela 4 – Base de dados com a preparação 2

	A1	A2	A3	A5	A7	A8	CLASS
0	False	9.6	94	True	False	0.294057	c0
1	False	5.2	74	False	True	0.204914	c0
2	False	2.2	43	True	True	0.655174	c2
3	True	10.0	63	True	False	0.308093	c0
4	True	2.0	25	False	True	0.836315	c1
..	...	...	...	...	...	...	...
995	True	2.9	60	False	False	0.132663	c0
996	False	2.6	69	True	True	0.714479	c2
997	False	3.6	41	True	True	0.612487	c2
998	True	1.1	72	True	False	0.794731	c0
999	False	1.8	76	True	True	0.103138	c2

Fonte: Elaborada pelo autor.

2.4 Preparação 3

A terceira preparação consistirá no conjunto da preparação 1 e 2, aplicando a redução dos atributos e a transformação dos atributos contínuos (Tabela 5).

Tabela 5 – Base de dados com a preparação 2

	A1	A2	A3	A5	A7	A8	CLASS
0	False	0.96	0.94	True	False	0.294057	c0
1	False	0.52	0.74	False	True	0.204914	c0
2	False	0.22	0.43	True	True	0.655174	c2
3	True	1.00	0.63	True	False	0.308093	c0
4	True	0.20	0.25	False	True	0.836315	c1
..	...	...	...	...	...	...	...
995	True	0.29	0.60	False	False	0.132663	c0
996	False	0.26	0.69	True	True	0.714479	c2
997	False	0.36	0.41	True	True	0.612487	c2
998	True	0.11	0.72	True	False	0.794731	c0
999	False	0.18	0.76	True	True	0.103138	c2

Fonte: Elaborada pelo autor.

3 GRÁFICOS COMPARATIVOS COM OS ALGORITMOS

3.1 Individual - *Perceptron* Multicamadas

A *perceptron* multicamadas (*Multilayer Perceptron* - (MLP)) é uma evolução da *perceptron* que, de acordo com Grus (2016) é a rede neural mais simples que resolve apenas problemas linearmente separáveis. Essa evolução da rede neural é dada por um maior processamento das entradas, tendo várias camadas ocultas (*hidden layer*) apresentando vários neurônios independentes, com funções de ativação e pesos de entradas diferentes chegando até eles de modo em que todas entradas participam de todos os neurônios. Dessa forma, a saída desse aprendizado de máquina é dada por a combinação de vários perceptrons, cada um com um peso, gerando inúmeras camadas ocultas que conseguem classificar problemas não linearmente separáveis.

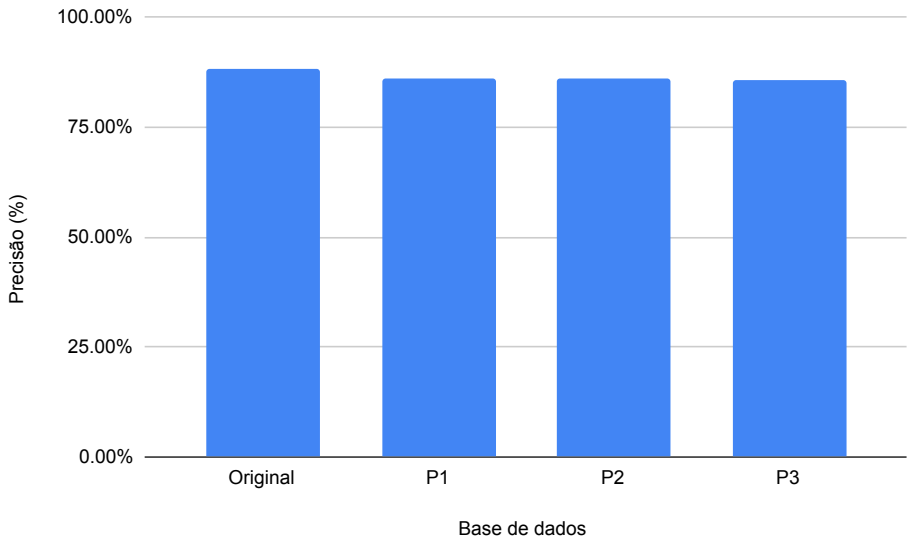

Figura 3 – Gráfico Perceptron Multicamadas - Precisão.
Fonte: Elaborada pelo autor.

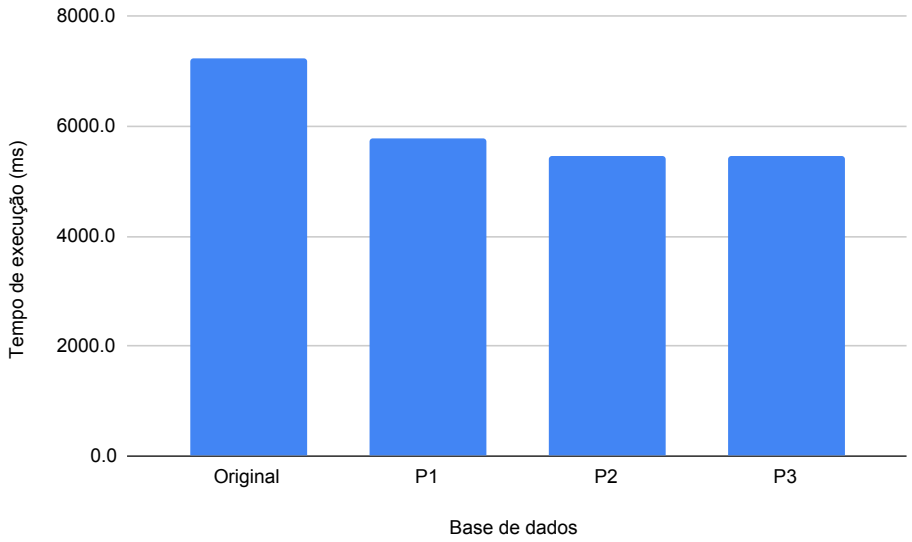

Figura 4 – Gráfico Perceptron Multicamadas - Tempo de execução.
Fonte: Elaborada pelo autor.

3.2 Individual - Floresta Aleatória

O algoritmo Floresta Aleatória (*Random Forest*) é um aprendizado que consegue trabalhar quanto com classificação, tanto regressão. Ele é uma evolução da árvore de decisão que evita o possível sobreajuste, da forma em que ela pode construir várias árvore de decisão com diferentes seleções de atributos (ao invés da seleção a partir do cálculo de impureza) para deixá-las escolher como classificar as entradas (GRUS, 2016).

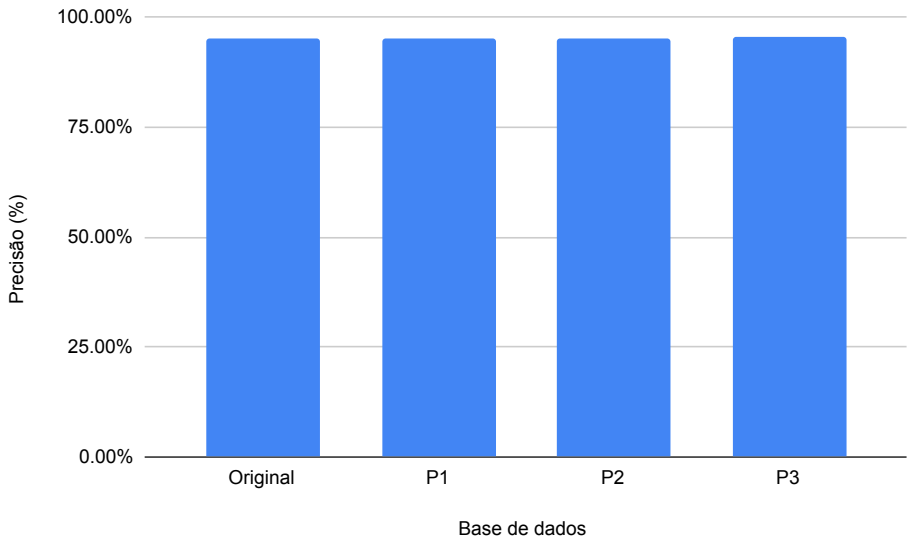

Figura 5 – Gráfico Floresta Artificial - Precisão.
Fonte: Elaborada pelo autor.

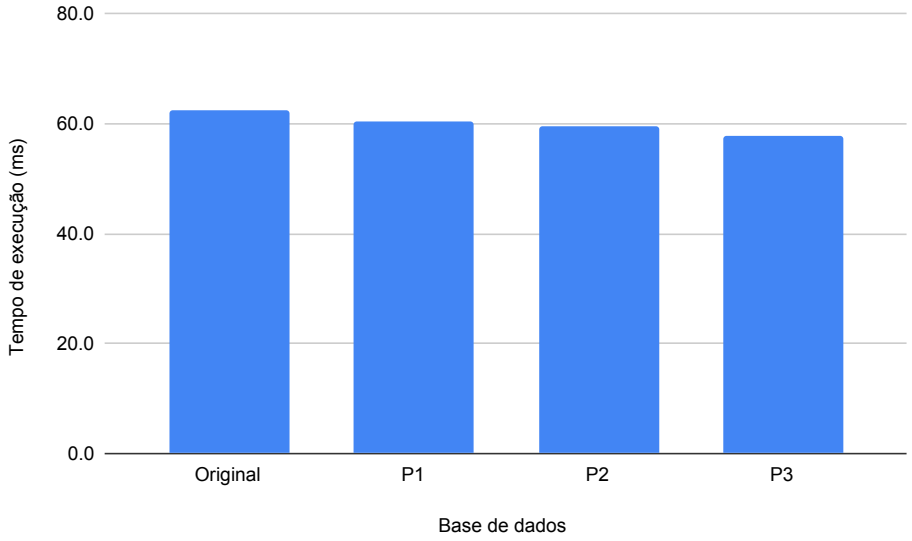

Figura 6 – Gráfico Floresta Artificial - Tempo de execução.
Fonte: Elaborada pelo autor.

O método *bootstrap dataset* cria um subconjunto de amostras aleatórias a partir da base de dados, depois é selecionado N características aleatoriamente para compor um árvore, é escolhido o melhor atributo do conjunto dos N, sendo ele o atributo que irá compor a primeira condição da árvore. Essas N características são novamente escolhidas aleatoriamente até passar por todos atributos. Toda essa tarefa compõe uma árvore aleatória, podendo ser feita várias vezes com diferentes amostras para compor toda a floresta.

3.3 Individual - Gradiente Descendente Estocástico

Esse algoritmo veio como uma variação do algoritmo Gradiente Descendente que consiste encontrar os valores dos vários parâmetros de forma iterativa, que minimizam determinada função de interesse. O Gradiente Descendente Estocástico (*Stochastic Gradient Descent* - SGD), por atualizar os parâmetros a cada registro em vez de passar por todas as amostras do seu conjunto de treinamento, apresenta um custo computacional menor e uma melhor classificação para alguns casos de base de dados.

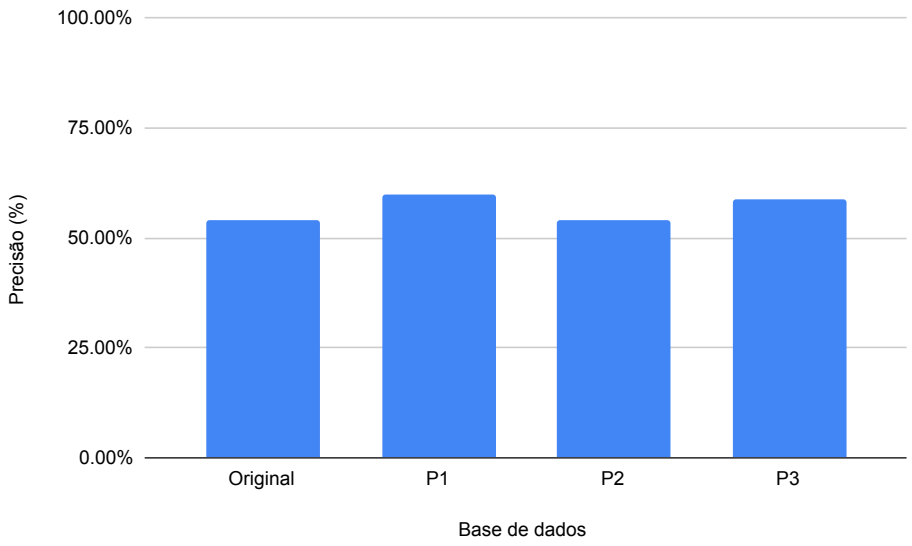

Figura 7 – Gráfico Gradiente Descendente Estocástico - Precisão.
Fonte: Elaborada pelo autor.

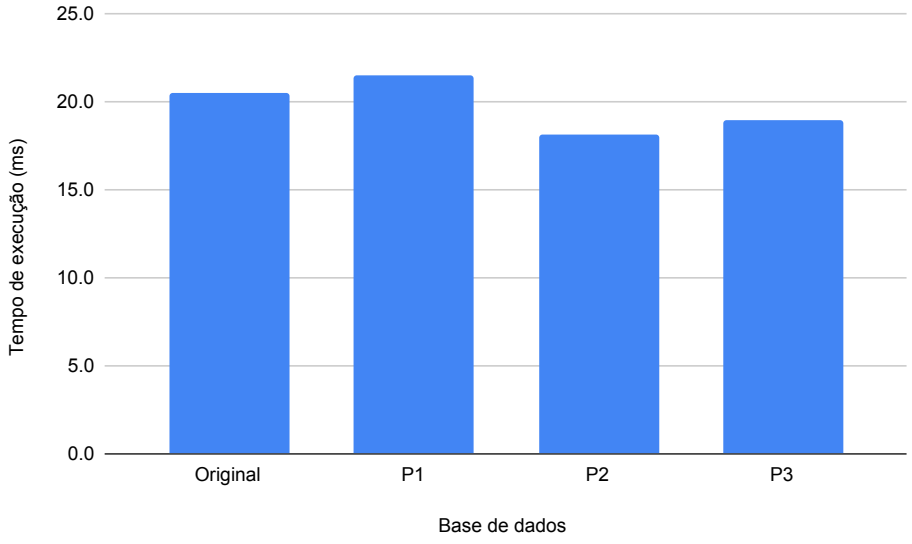

Figura 8 – Gráfico Gradiente Descendente Estocástico - Tempo de execução.
Fonte: Elaborada pelo autor.

3.4 Geral

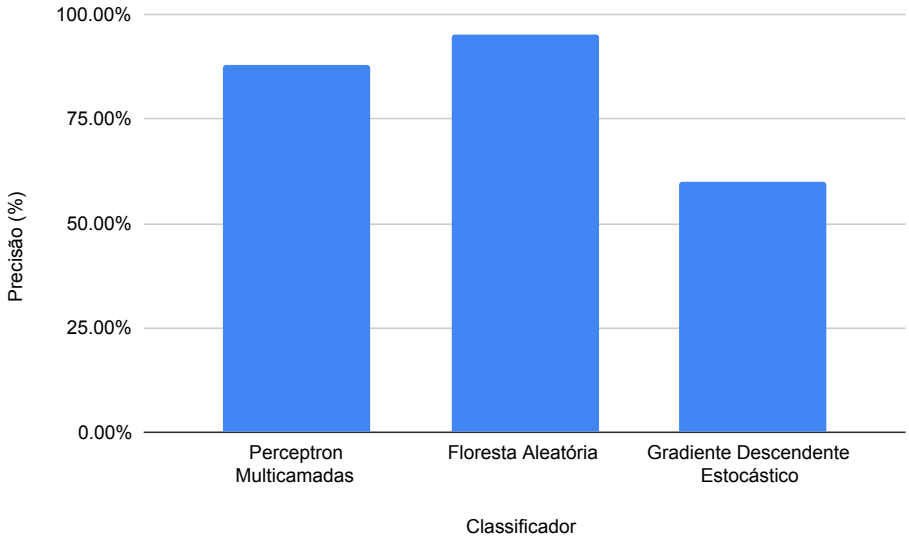

Figura 9 – Gráfico Geral - Maior precisão.
Fonte: Elaborada pelo autor.

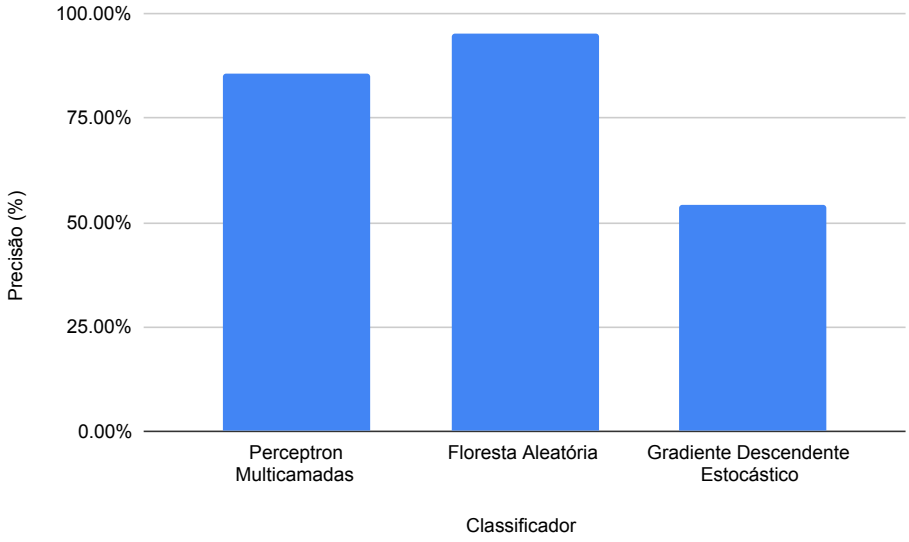

Figura 10 – Gráfico Geral - Menor precisão.
Fonte: Elaborada pelo autor.

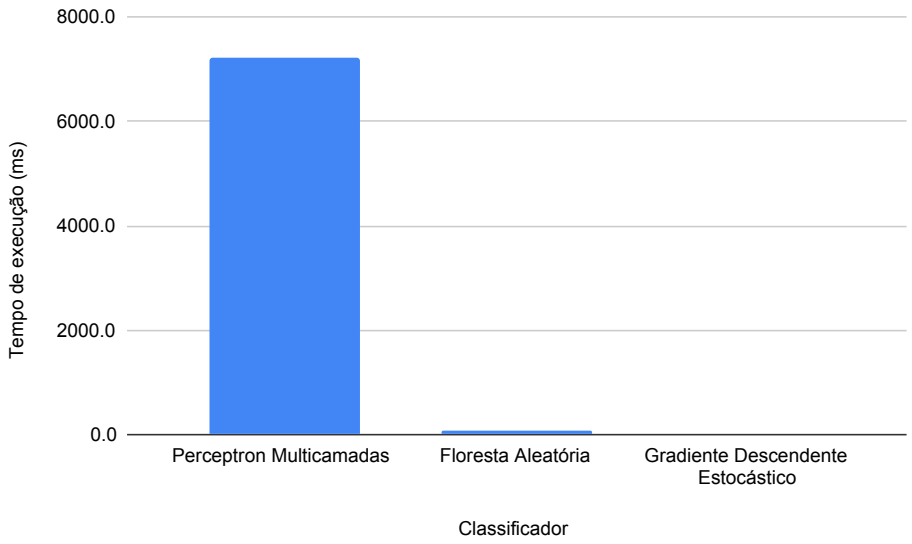

Figura 11 – Gráfico Geral - Maior tempo de execução.
Fonte: Elaborada pelo autor.

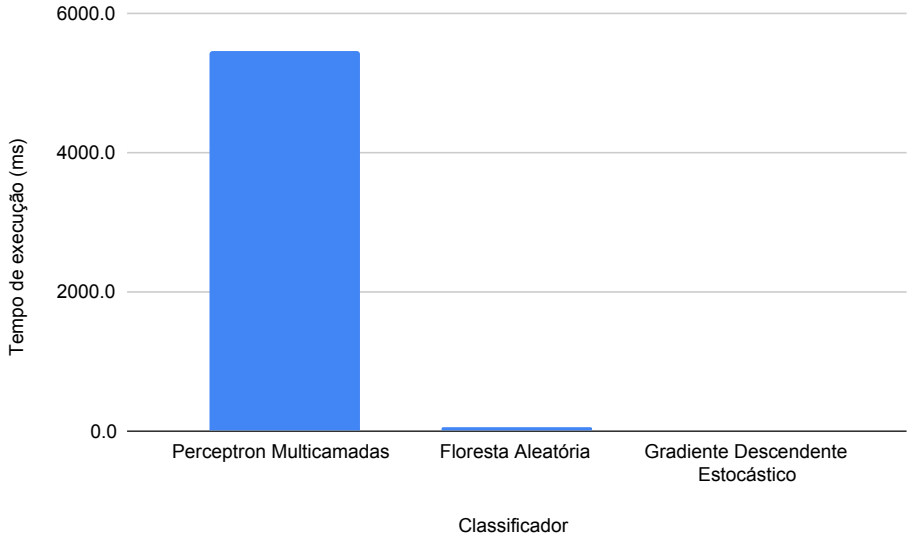

Figura 12 – Gráfico Geral - Menor tempo de execução.
Fonte: Elaborada pelo autor.

4 CONCLUSÃO

Os parâmetros colocados nos algoritmos foram ajustados de maneira que a precisão de acertos das instâncias fossem a maior possível, podendo aumentar o tempo de execução do algoritmo, como no caso do *Perceptron* Multicamadas.

As bases preparadas obtiveram um tempo de resposta menor comparado a base original, mas, em questão de precisão, a diferença foi mínima, concluindo que os algoritmos já estão preparados para aqueles tipos de problemas filtrados pelas preparações, sendo apenas executado mais rápido pelos classificadores. A preparação 2 apresentou complicações significativas para a

precisão do Gradiente Descendente Estocástico.

Para a base de dados do trabalho, a floresta aleatória apresentou os melhores resultados. Se compararmos o conjunto de precisão e tempo de execução dela com os outros algoritmos, ela sobressai melhor significativamente.

REFERÊNCIAS

BONIFÁCIO, F. N. Comparação entre as redes neurais artificiais MLP, RBF e LVQ na classificação de dados. **Paraná: Universidade Estadual do Oeste do Paraná**, 2010.

FISHER, D.; HAPANYENGWI, G. Database management and analysis tools of machine induction. **Journal of Intelligent Information Systems**, Springer, v. 2, n. 1, p. 5–38, 1993.

GRUS, J. **Data Science do zero: Primeiras regras com o Python**. 1. ed.: Alta Books, 2016.

NONG, Y. **The handbook of data mining**. 1. ed.: Lawrence Erlbaum Associates, Publishers, 2003. (Human factors and ergonomics).

ZHANG, G. P. Neural networks for classification: a survey. **Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, IEEE, v. 30, n. 4, p. 451–462, 2000.

APÊNDICE A - Código de desenvolvimento

```
#imports
import time
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import neural_network
from sklearn import ensemble
from sklearn import linear_model
import pandas as pd



#Teste dos classificadores
def classify(classifier):
    runtime = []
    score = []


    #base original
    for i in range(15):
        start = time.time()
        classifier.fit(X_train, y_train)
        end = time.time()
        runtime.append(end-start)
        score.append(classifier.score(X_test, y_test))
    print((sum(score)/len(score)), (sum(runtime)/len(runtime)))
    runtime.clear()
    score.clear()


    #base preparada 1
    for i in range(15):
        start = time.time()
        classifier.fit(X_train, y_train)
        end = time.time()
        runtime.append(end-start)
        score.append(classifier.score(X1_test, y1_test))
    print((sum(score)/len(score)), (sum(runtime)/len(runtime)))
    runtime.clear()
    score.clear()


    #base preparada 2
    for i in range(15):
        start = time.time()
        classifier.fit(X_train, y_train)
        end = time.time()
```

```
47          runtime.append(end-start)
48          score.append(classifier.score(X2_test, y2_test))
49      print((sum(score)/len(score)), (sum(runtime)/len(runtime)))
50      runtime.clear()
51      score.clear()
52
53
54      #base preparada 3
55      for i in range(15):
56          start = time.time()
57          classifier.fit(X_train, y_train)
58          end = time.time()
59          runtime.append(end-start)
60          score.append(classifier.score(X3_test, y3_test))
61      print((sum(score)/len(score)), (sum(runtime)/len(runtime)))
62
63
64
65  #base original
66  df = pd.read_csv('classifica.csv')
67  df = df.drop(columns=['ID'])
68
69  X_train, X_test, y_train, y_test = train_test_split(df.drop(columns = ['
        CLASS']), df['CLASS'], test_size = 0.1, random_state=5)
70
71  print(df)
72
73
74
75  #Preparação 1
76  df_p1 = df.copy()
77
78  df_p1['A2'] = df_p1['A2']/10
79  df_p1['A3'] = df_p1['A3']/100
80
81  X1_train, X1_test, y1_train, y1_test = train_test_split(df.drop(columns = ['
        CLASS']), df['CLASS'], test_size = 0.1, random_state=5)
82
83  print (df_p1)
84
85
86
87  #Preparação 2
88  df_p2 = df.copy()
89
90  df_p2 = df_p2.drop(columns=['A4','A6'])
91
92  X2_train, X2_test, y2_train, y2_test = train_test_split(df.drop(columns = ['
```

```
    CLASS']), df['CLASS'], test_size = 0.1, random_state=5)
93
94 print (df_p2)
95
96
97
98 #Preparação 3
99 df_p3 = df.copy()
100
101 df_p3 = df_p3.drop(columns=['A4','A6'])
102 df_p3['A2'] = df_p3['A2']/10
103 df_p3['A3'] = df_p3['A3']/100
104
105 X3_train, X3_test, y3_train, y3_test = train_test_split(df.drop(columns = ['
    CLASS']), df['CLASS'], test_size = 0.1, random_state=5)
106
107 print(df_p3)
108
109
110
111 #Perceptron Multicamadas
112
113 perceptron = neural_network.MLPClassifier(max_iter=12000, hidden_layer_sizes
    =50, solver = 'lbfgs')
114 classify(perceptron)
115
116
117 #Floresta Aleatória
118
119 rf = ensemble.RandomForestClassifier(n_estimators = 25, criterion= 'gini',
    max_features=None)
120 classify(rf)
121
122
123 #Gradiente Descendente Estocástico
124
125 sgd = linear_model.SGDClassifier(loss = 'modified_huber', penalty='
    elasticnet', max_iter = 12000)
126 classify(sgd)
```