UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE RUSSAS

RELATÓRIO DE TESTES DE ALGORITMOS DE CLASSIFICAÇÃO NA BASE DE DADOS PIMA INDIANS DIABETES

Maria Clara de Abreu Silva
Maria Elanne Mendes Rodrigues

Novembro de 2019

1. Introdução

Algoritmos de classificação são utilizados para mapear elementos semelhantes em categorias específicas. Neste trabalho, serão abordados os seguintes algoritmos: *kNN*, *Random Forest*, *Naive Bayes*, *Decision Tree* e *Perceptron*, cujos testes foram feitos na plataforma Jupyter Notebook e a classificação individual de cada classificador fora feita baseada na sua acurácia.

2. Descrição de experimentos

2.1. Base de dados

A base de dados escolhida para os experimentos é a Pima Indians Diabetes, o qual é composto por informações referentes à mulheres com pelo menos 21 anos de idade, cuja origem é indígena dos povos Pima. Nela, estão inseridas 768 instâncias (número de linhas) com 9 atributos cada, sendo que o último, informa se a pessoa possui a doença ou não.

Todos os atributos são preenchidos com valor numérico e correspondem a:

- Número de vezes que a pessoa ficou grávida - *preg*;
- Concentração de glicose no plasma em um período de duas horas em um teste oral de tolerância à glicose - *plas*;
- Pressão arterial diastólica (mm Hg) - *pres*;
- Espessura da dobra da pele do tríceps (mm) - *skin;*
- Insulina sérica de 2 horas (mu U / ml) - *test;*
- Índice de massa corporal - *mass;*
- Histórico de diabetes na família - *pedi;*
- Idade - *age;*
- Situação (0 - não possui a doença; 1 - possui a doença) - *class;*

2.2. Algoritmos de classificação

Classificação é um método que aprende uma função, em termos matemáticos, com valores de atributos em seu domínio e valores de classe em sua imagem. Em outras palavras, dado um conjunto de valores de atributos, tal função determina um valor de classe para esse conjunto. Essa função é também conhecida como modelo de classificação (SILVA, 2007).

Essa classificação é feita através de algoritmos de aprendizagem. Esses algoritmos conseguem “criar conhecimento” sobre o domínio tratado através de dados de entrada. Quando esses dados de entrada já possuem suas classes atribuídas dizemos que essa é uma aprendizagem supervisionada. Dessa forma, o algoritmo consegue criar uma relação entre as características de cada entrada com a sua classe final, gerando um “conhecimento” para fazer previsões sobre entradas que ainda não possuem uma classe definida. Esses dados de entrada já classificados são chamados de base de treino enquanto que os documentos que

serão classificados posteriormente ao treino são denominados como a base de teste (SANTOS, 2013).

Quando a base de treino não possui uma classificação pré-definida são utilizados algoritmos de aprendizagem não-supervisionada. Nesse caso, os algoritmos tentam agrupar entradas com características semelhantes e classificá-los por grupos. Esse tipo de classificação é chamada de clusterização (SANTOS, 2013).

2.2.1. Random Forest

O algoritmo *Random Forest* trata-se de um classificador que faz uso do método de árvores de decisão criada por Breiman (2001) possibilitando a mineração dos dados passados a mesma. Esta técnica possui uma ideia um pouco diferente dos algoritmos de árvores de decisão, enquanto uma árvore possui o objetivo de construção total de uma estrutura a partir de uma base de dados, o *Random Forest* tem o objetivo de efetuar a criação de várias árvores de decisão usando um subconjunto de atributos selecionados aleatoriamente a partir do conjunto original, contendo todos os atributos e que estes possuem um tipo de amostragem chamado de bootstrap, a qual é do tipo com reposição, possibilitando assim melhor análise dos dados. (NETO, 2014).

Figura 1 - Uso do algoritmo *Random Forest*

```
In [1]: import pandas as pd
        from sklearn.model_selection import train_test_split #importando a biblioteca p/ separação das instâncias de treino e teste
        from sklearn.ensemble import RandomForestClassifier #importando a biblioteca que será utilizada no algoritmo
        import sklearn.metrics as metrics #importando a bib para calcular a acurácia

In [2]: df = pd.read_csv("diabetes.csv") #carrega arquivo do dataset

In [3]: x = df.values[:,:-1] #fazendo com que o x receba os valores até uma coluna antes do resultado
        y = df.values[:,-1] #fazendo com que o y receba os valores de todas as colunas, inclusive da coluna de resultado

In [4]: X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.33)

In [5]: model = RandomForestClassifier() #informando que o meu modelo receberá um resultado advindo do algortimo Random Forest

In [6]: model.fit(X_train, y_train)

        C:\Users\mclar\Anaconda2019\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_estimators
        will change from 10 in version 0.20 to 100 in 0.22.
          "10 in version 0.20 to 100 in 0.22.", FutureWarning)

Out[6]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                               max_depth=None, max_features='auto', max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=10,
                               n_jobs=None, oob_score=False, random_state=None,
                               verbose=0, warm_start=False)

In [7]: y_pred = model.predict(X_test)

In [8]: metrics.accuracy_score(y_test, y_pred)

Out[8]: 0.7362204724409449
```

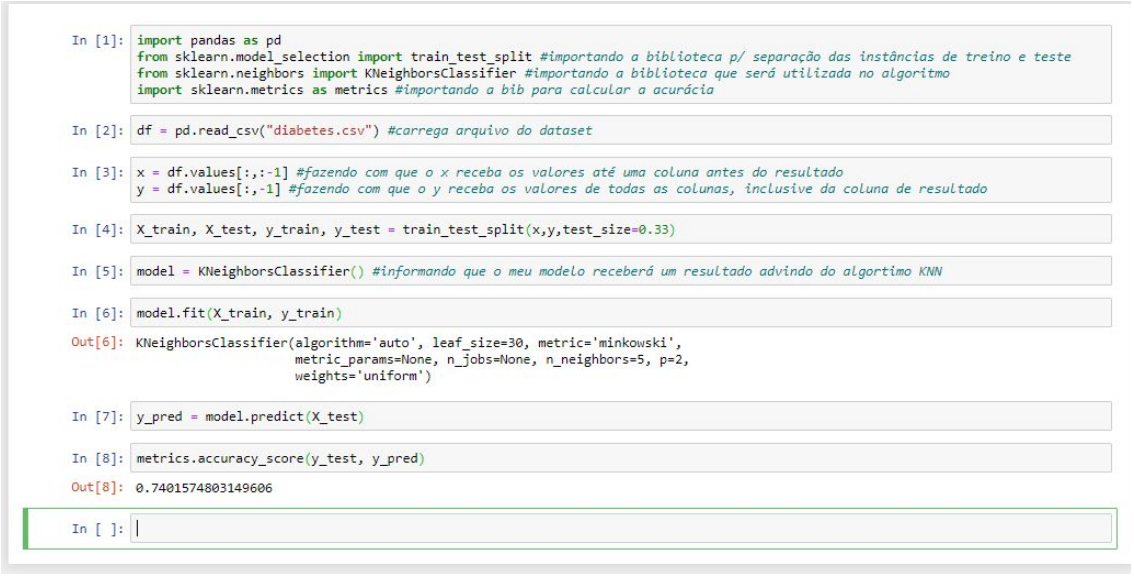
Fonte: Elaboração própria

2.2.2. kNN

O algoritmo *k-Nearest Neighbor (kNN)* é um algoritmo de aprendizado supervisionado do tipo *lazy*. A ideia geral desse algoritmo consiste em encontrar os *k* exemplos rotulados mais próximos do exemplo não classificado e, com base no

rótulo desses exemplos mais próximos, é tomada a decisão relativa à classe do exemplo não rotulado. Os algoritmos da família *kNN* requerem pouco esforço durante a etapa de treinamento. Em contrapartida, o custo computacional para rotular um novo exemplo é relativamente alto, pois, no pior dos casos, esse exemplo será comparado com todos os exemplos contidos no conjunto de treinamento (FERRERO, 2009).

Figura 2 - Uso do algoritmo *kNN*



```
In [1]: import pandas as pd
        from sklearn.model_selection import train_test_split #importando a biblioteca p/ separação das instâncias de treino e teste
        from sklearn.neighbors import KNeighborsClassifier #importando a biblioteca que será utilizada no algoritmo
        import sklearn.metrics as metrics #importando a bib para calcular a acurácia

In [2]: df = pd.read_csv("diabetes.csv") #carrega arquivo do dataset

In [3]: x = df.values[:,:-1] #fazendo com que o x receba os valores até uma coluna antes do resultado
        y = df.values[:,-1] #fazendo com que o y receba os valores de todas as colunas, inclusive da coluna de resultado

In [4]: X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.33)

In [5]: model = KNeighborsClassifier() #informando que o meu modelo receberá um resultado advindo do algortimo KNN

In [6]: model.fit(X_train, y_train)
Out[6]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                             weights='uniform')

In [7]: y_pred = model.predict(X_test)

In [8]: metrics.accuracy_score(y_test, y_pred)
Out[8]: 0.7401574803149606

In [ ]:
```

Fonte: Elaboração própria

2.2.3. Decision Tree

O algoritmo *Decision Tree* ou *Árvore de Decisão* é uma representação simples de um classificador utilizada por diversos sistemas de aprendizado de máquina. Uma árvore de decisão é induzida a partir de um conjunto de exemplos de treinamento onde as classes são previamente conhecidas.

A estrutura da árvore é organizada de tal forma que:

a) cada nó interno (não-folha) é rotulado com o nome de um dos atributos previsores;
b) os ramos (ou arestas) saindo de um nó interno são rotulados com valores do atributo naquele nó;
c) cada folha é rotulada com uma classe, a qual é a classe prevista para exemplos que pertençam àquele nó folha.

O processo de classificação de um exemplo ocorre fazendo aquele exemplo “caminhar” pela árvore, a partir do nó raiz, procurando percorrer os arcos que unem os nós, de acordo com as condições que estes mesmos arcos representam. Ao atingir um nó folha, a classe que rotula aquele nó folha é atribuída àquele exemplo (CARVALHO, 2005).

Figura 3 - Uso do algoritmo *Decision Tree*

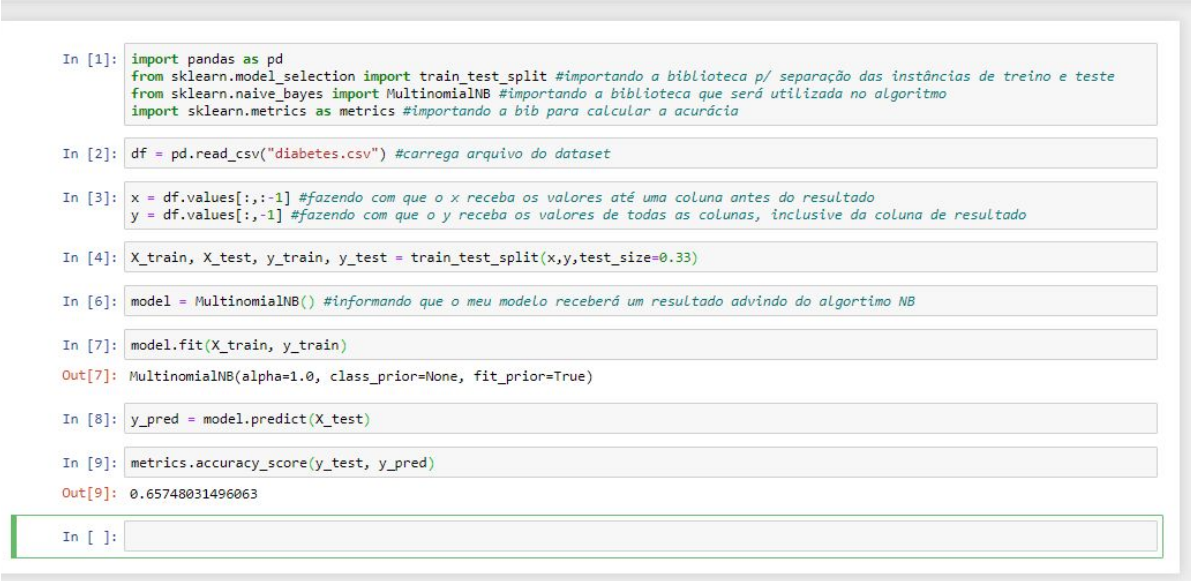
```
In [1]: import pandas as pd
        from sklearn.model_selection import train_test_split #importando a biblioteca p/ separação das instâncias de treino e teste
        from sklearn.tree import DecisionTreeClassifier #importando a biblioteca que será utilizada no algoritmo
        import sklearn.metrics as metrics #importando a bib para calcular a acurácia

In [2]: df = pd.read_csv("diabetes.csv") #carrega arquivo do dataset

In [3]: x = df.values[:,:-1] #fazendo com que o x receba os valores até uma coluna antes do resultado
        y = df.values[:,-1] #fazendo com que o y receba os valores de todas as colunas, inclusive da coluna de resultado

In [4]: X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.33)

In [5]: model = DecisionTreeClassifier() #informando que o meu modelo receberá um resultado advindo do algortimo Decision Tree

In [6]: model.fit(X_train, y_train)

Out[6]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                               max_features=None, max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, presort=False,
                               random_state=None, splitter='best')

In [7]: y_pred = model.predict(X_test)

In [8]: metrics.accuracy_score(y_test, y_pred)

Out[8]: 0.6811023622047244
```

Fonte: Elaboração própria

2.2.4. Naive Bayes

O algoritmo *Naive Bayes* é um classificador probabilístico simples que calcula um conjunto de probabilidades contando a frequência e as combinações de valores em um determinado conjunto de dados. O algoritmo usa o teorema de *Bayes* e assume que todos os atributos são independentes, dado o valor da variável de classe. Essa suposição de independência condicional raramente é verdadeira em aplicações do mundo real, daí a caracterização como ingênua, mas o algoritmo tende a ter um bom desempenho e aprender rapidamente em vários problemas de classificação supervisionada. Essa "ingenuidade" permite que o algoritmo construa facilmente classificações a partir de grandes conjuntos de dados sem recorrer a esquemas de estimativa de parâmetros iterativos complicados (DIMITOGLOU; ADAMS & JIM, 2012).

Figura 5 - Uso do algoritmo *Naive Bayes*



```
In [1]: import pandas as pd
        from sklearn.model_selection import train_test_split #importando a biblioteca p/ separação das instâncias de treino e teste
        from sklearn.naive_bayes import MultinomialNB #importando a biblioteca que será utilizada no algoritmo
        import sklearn.metrics as metrics #importando a bib para calcular a acurácia

In [2]: df = pd.read_csv("diabetes.csv") #carrega arquivo do dataset

In [3]: x = df.values[:,:-1] #fazendo com que o x receba os valores até uma coluna antes do resultado
        y = df.values[:,-1] #fazendo com que o y receba os valores de todas as colunas, inclusive da coluna de resultado

In [4]: X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.33)

In [6]: model = MultinomialNB() #informando que o meu modelo receberá um resultado advindo do algortimo NB

In [7]: model.fit(X_train, y_train)
Out[7]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

In [8]: y_pred = model.predict(X_test)

In [9]: metrics.accuracy_score(y_test, y_pred)
Out[9]: 0.65748031496063

In [ ]:
```

Fonte: Elaboração própria

2.2.5. Perceptron

O *Perceptron* consiste em uma única camada de neurônios com pesos sinápticos e bias ajustáveis. Se os padrões de entrada forem linearmente separáveis, o algoritmo de treinamento do *Perceptron* possui convergência garantida, ou seja, é capaz de encontrar um conjunto de pesos que classifica corretamente os dados. Os pesos dos neurônios que compõem o *Perceptron* serão tais que as superfícies de decisão produzidas pela rede neural estarão apropriadamente posicionadas no espaço (CASTRO F. & CASTRO M., 2001).

Figura 5 - Uso do algoritmo *Perceptron*

```
In [11]: import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import Perceptron
         import sklearn.metrics as metrics

In [2]: df = pd.read_csv("diabetes.csv")

In [3]: x = df.values[:,:-1]
        y = df.values[:,-1]

In [5]: X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.33)

In [7]: model = Perceptron()

In [8]: model.fit(X_train, y_train)

Out[8]: Perceptron(alpha=0.0001, class_weight=None, early_stopping=False, eta0=1.0,
                   fit_intercept=True, max_iter=1000, n_iter_no_change=5, n_jobs=None,
                   penalty=None, random_state=0, shuffle=True, tol=0.001,
                   validation_fraction=0.1, verbose=0, warm_start=False)

In [9]: y_pred = model.predict(X_test)

In [12]: metrics.accuracy_score(y_test, y_pred)

Out[12]: 0.6535433070866141
```

Fonte: Elaboração própria

3. Resultados

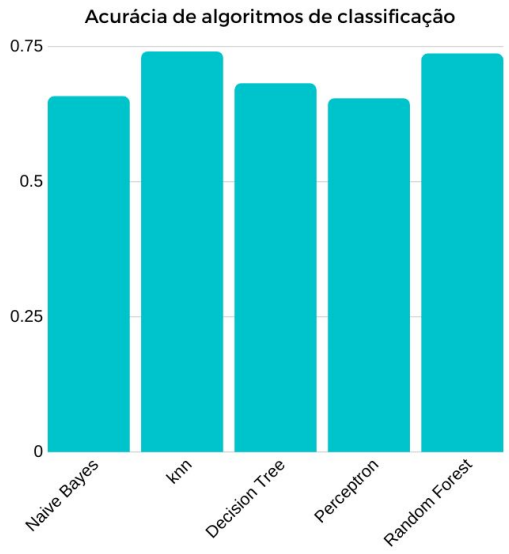
Como resultado obtido, observa-se que o algoritmo *knn* obteve a maior acurácia, seguido pelo *Random Forest.*

Tabela 1 - Tabela com acurácia de cada algoritmo

Algoritmo	Acurácia
Naive Bayes	0.65748031496063
kNN	0.7401574803149606
Decision Tree	0.6811023622047244
Perceptron	0.6535433070866141
Random Forest	0.7362204724409449

Fonte: Elaboração própria

Figura 6 - Gráfico comparativo da acurácia de cada algoritmo



Fonte: Elaboração própria

Referências

SILVA, J. **Algoritmos de classificação baseados em análise formal de conceitos.** Master's thesis, Universidade Federal de Minas Gerais, 2007.

SANTOS, F. L. **Mineração de opinião em textos opinativos utilizando algoritmos de classificação**. 2013.

CASTRO, F. C. C.; CASTRO, M. C. F. **Redes neurais artificiais**. DCA/FEEC/Unicamp, 2001.

FERRERO, C. A. **Algoritmo kNN para previsão de dados temporais: funções de previsão e critérios de seleção de vizinhos próximos aplicados a variáveis ambientais em limnologia**. 2009. Tese de Doutorado. Universidade de São Paulo.

CARVALHO, D. R. **Árvore de decisão/algoritmo genético para tratar o problema de pequenos disjuntos em classificação de dados**. Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil. Doctor Thesis. 162pp, 2005.

DIMITOGLOU, G.; ADAMS, J. A.; JIM, C. M. **Comparison of the C4. 5 and a Naïve Bayes classifier for the prediction of lung cancer survivability**. arXiv preprint arXiv:1206.1121, 2012.

NETO, C. D. G. **Potencial de técnicas de mineração de dados para o mapeamento de áreas cafeeiras**. INPE, São José dos Campos, 2014.