Inteligência Artificial

Thiago Henrique Leite da Silva, RA: 139920

AULA10: Exercício prático aprendizado de máquina

1) Testar o SVM e uma MLP para o dataset IRIS: https://www.kaggle.com/uciml/iris.

Separe aleatoriamente 70% dos dados para treino e 30% para teste e reporte com um print da saída qual a acurácia do algoritmo (número de acertos).

SVM

A máquina de vetores de suporte é um algoritmo de aprendizagem de máquina, supervisionado, que serve tanto para classificação quanto para regressão, com foco principal em treinamento e classificação de um determinado dataset. O que iremos fazer é visualizar a aplicação deste algoritmo na prática no treinamento do dataset Iris, e posteriormente veremos sua acurácia na assertividade dos casos teste.

O código que utilizei não é de própria autoria, e sim da cientista de dados Kizzy, integrante do canal no YouTube “Programa Dinâmica”; o código foi implementado em Python, algo que achei bem interessante também por ter sido meu primeiro contato com a linguagem que fugiu do básico que os cursos em vídeo nos ensinam. Encontrei vários recursos que me lembravam o R visto em probabilidade e estatística, além de ser um código bem legível com alguns detalhes de sintaxe parecido com Ruby, linguagem que tenho um pouco mais de familiaridade.

Partindo para o que interessa, vamos falar sobre a **implementação**:

Primeiramente importamos as bibliotecas necessárias para a parte inicial do código:

```
from sklearn import datasets
import pandas as pd
import seaborn as sns
```

Carregamos o dataset Iris que já pertence a biblioteca datasets, transformamos ele em um ‘pandas dataframe’, para conseguirmos ver o dataset em forma de gráficos e frames. Também adicionamos um label que será o valor numérico da classe. (setosa=0, versicolor=1, virginica=2)

```
iris = datasets.load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['label'] = iris.target
iris_df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
```

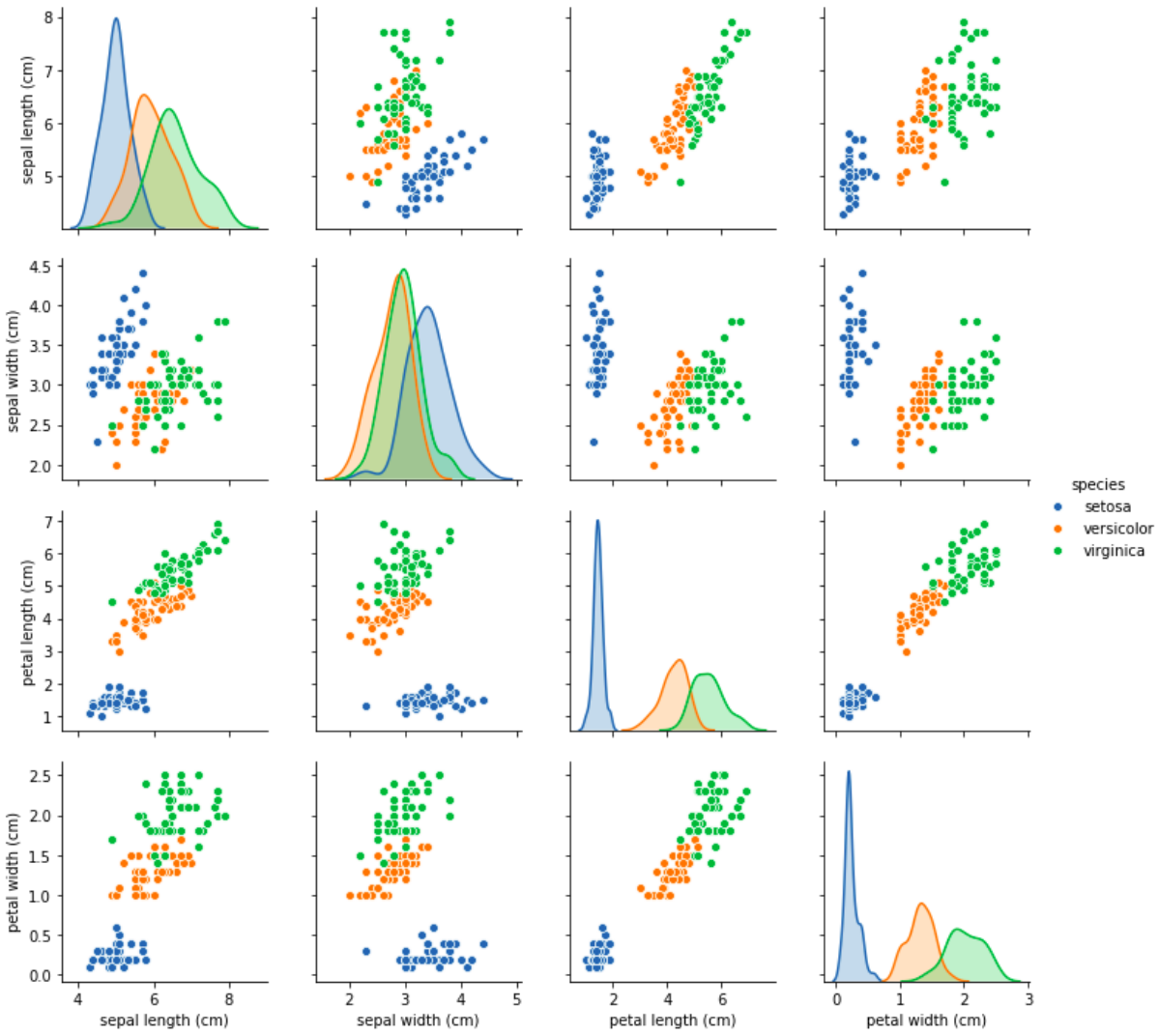
Ao imprimirmos as 5 primeiras linhas de nosso dataset obtemos o seguinte resultado:

```
iris_df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	label	species
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

A partir daí, podemos plotar em pares as características (colunas) do nosso dataset e assim visualizar se eles são linearmente separáveis.

```
sns.pairplot(iris_df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
        'petal width (cm)','species']], hue='species')
```



Depois a programadora mostra o conceito de margens, na primeira linha nós pegamos todos os index de íris versicolor, e na segunda linha salvamos em uma variável um dataset parcial sem aqueles index, ou seja, somente com íris setosa e virginica.

```
versicolor_index = iris_df[iris_df['species'] == 'versicolor'].index
partial_iris_df = iris_df.drop(versicolor_index).reset_index()
```

```
Int64Index([50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,
            67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83,
            84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99],
           dtype='int64')
```

	index	sepal length (cm)	sepal width (cm)	petal length (cm)	\
0	0	5.1	3.5	1.4	
1	1	4.9	3.0	1.4	
2	2	4.7	3.2	1.3	
3	3	4.6	3.1	1.5	
4	4	5.0	3.6	1.4	
..	...	...	...	...	
95	145	6.7	3.0	5.2	
96	146	6.3	2.5	5.0	
97	147	6.5	3.0	5.2	
98	148	6.2	3.4	5.4	
99	149	5.9	3.0	5.1	

	petal width (cm)	label	species
0	0.2	0	setosa
1	0.2	0	setosa
2	0.2	0	setosa
3	0.2	0	setosa
4	0.2	0	setosa
..	...	...	...
95	2.3	2	virginica
96	1.9	2	virginica
97	2.0	2	virginica
98	2.3	2	virginica
99	1.8	2	virginica

```
[100 rows x 7 columns]
```

Agora já podemos partir para a classificação do dataset, vamos importar as bibliotecas necessárias:

```
from sklearn import svm
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

Iremos salvar na variável X a parte principal do nosso dataset, que serão as 4 colunas de com os atributos de nossos indivíduos (íris), e em y salvamos os labels das espécies.

```
X,y = load_iris(return_X_y=True)
```

X é dessa forma:

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
```

Y é dessa forma:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Posteriormente, vamos fazer a divisão entre treinamento e teste com o método train_test_split(), sendo o 70% de treino e 30% de teste.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=3)
```

Então, instanciamos um classificador com kernel linear, ou seja, estamos buscando uma reta para realizar a divisão das classes, e o valor padrão de C, que indica se a margem é rígida ou suave.

```
C = 1.0  # SVM regularization parameter
clf = svm.SVC(kernel='linear', C=C)
```

Depois fazemos o fit com os dados de treinamento para que o algoritmo seja treinado.

```
clf.fit(X_train, y_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

Após o treinamento, podemos fazer a predição do nosso conjunto de teste. Salvamos o resultado em uma variável.

```
y_pred = clf.predict(X_test)
```

O comando acima nos retorna à classificação de cada um dos indivíduos de teste.

```
y_pred
```

```
array([0, 0, 0, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 2, 0, 2, 2, 2, 0, 2, 2,
       2, 1, 0, 2, 2, 1, 1, 1, 0, 0, 2, 1, 0, 0, 1, 0, 2, 1, 2, 1, 0, 0,
       2])
```

Podemos imprimir a saída que esperávamos, definida lá em cima, para comparar visualmente os reultados.

```
y_test
```

```
array([0, 0, 0, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 2, 0, 1, 2, 2, 0, 2, 2,
       2, 1, 0, 2, 2, 1, 1, 1, 0, 0, 2, 1, 0, 0, 1, 0, 2, 1, 2, 1, 0, 0,
       2])
```

Verificando o score do nosso modelo, obtivemos 97.77% de acerto nos casos de teste.

```
clf.score(X_test,y_test)
```

```
0.9777777777777777
```

Para uma melhor visualização do resultado, podemos utilizar a biblioteca metrics da sklearn para plotarmos um gráfico com os resultados um pouco mais detalhado. Assim podemos ver melhor a excelente performance do algoritmo para nosso dataset.

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        17
  versicolor       1.00      0.93      0.96        14
   virginica       0.93      1.00      0.97        14

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

Fonte: **https://github.com/programacaodinamica/machine-learning/blob/master/notebooks/M%C3%A1quinas%20de%20Vetores%20de%20Suporte%20(SVM).ipynb**

MLP

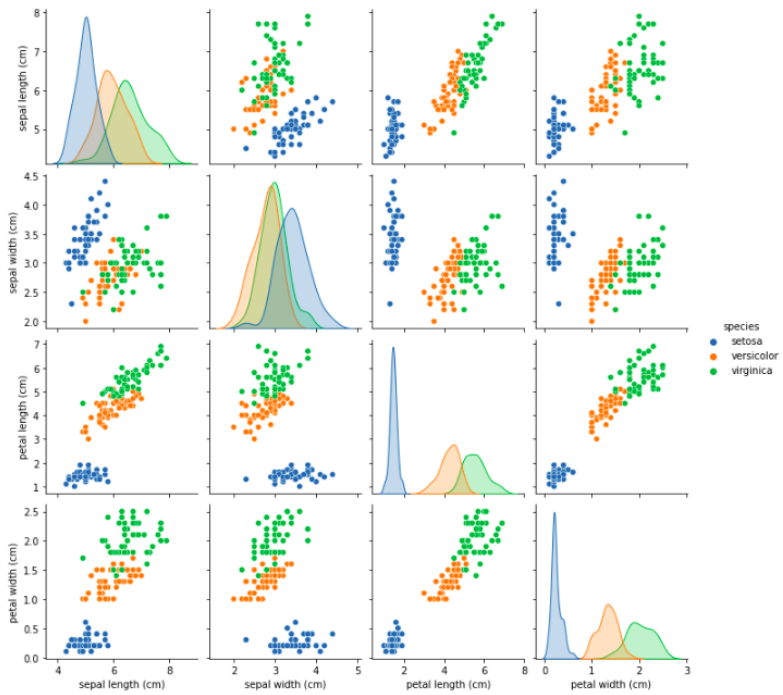
Para a execução do MLP para nosso dataset Íris, utilizei a base do algoritmo acima e apenas mudei o classificador, passei a utilizar a classe MLPClassifier da sklearn.neural_network.

Grande parte do código foi explicada acima, seguem também comentários nesse código:

```
from sklearn import datasets
import pandas as pd
import seaborn as sns

# Carregamos o dataset e adicionamos um label para cada uma das classes
iris = datasets.load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['label'] = iris.target
iris_df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)

# Imprime os 5 primeiros registros do nosso DF
iris_df.head()
```

```
# Plota os gráficos aos pares de nossos atributos

sns.pairplot(iris_df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
        'petal width (cm)','species']], hue='species')
```

```
<seaborn.axisgrid.PairGrid at 0x7fcfc37d8550>
```



Classificação:

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
# X recebe uma matriz com as features do dataset
# y recebe o label da classe de cada uma das espécies

X, y = load_iris(return_X_y=True)
```

```
# Separamos o dataset em casos de treino e casos de teste. 70% e 30%, respectivamente.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=3)
```

Aqui portanto chegamos no ponto chave desta implementação, o classificador. Desta vez utilizamos o MLPClassifier, que irá treinar nosso dataset, passamos como parâmetro:

- Random_state, que determina a geração de número aleatório para pesos e inicialização de polarização.
- Learnin_rate_init, taxa de aprendizagem inicial utilizada. Ela controla o tamanho do passo na atualização dos pesos.
- Max_iter, número máximo de iterações do algoritmo no treinamento.

A classe já utiliza implicitamente as variáveis X_train e y_train para sua execução.

```
# Treinamos o dataset com o algoritmo classificador MLP

clf = MLPClassifier(random_state = 1, learning_rate_init=0.003 ,max_iter=10000)
```

Após feito o treinamento, fazemos o fit e pegamos os resultados.

```
# Executamos o fit para os casos de treino

clf.fit(X_train, y_train)
```

```
MLPClassifier(learning_rate_init=0.003, max_iter=10000, random_state=1)
```

```
# Classificamos os casos de teste com base no resultado do treino

y_pred = clf.predict(X_test)
```

```
# Saída obtida

y_pred
```

```
array([0, 0, 0, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 2, 0, 2, 2, 2, 0, 2, 2,
       2, 1, 0, 2, 2, 1, 1, 1, 0, 0, 2, 1, 0, 0, 1, 0, 2, 2, 2, 1, 0, 0,
       2])
```

```
# Saída que era esperada

y_test
```

```
array([0, 0, 0, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 2, 0, 1, 2, 2, 0, 2, 2,
       2, 1, 0, 2, 2, 1, 1, 1, 0, 0, 2, 1, 0, 0, 1, 0, 2, 1, 2, 1, 0, 0,
       2])
```

```
# Calculamos o score dos acertos

clf.score(X_test,y_test)
```

```
0.9555555555555556
```

Por fim, chegamos ao resultado final do algoritmo:

```
# Plota gráfico com os resultados um pouco mais detalhados

from sklearn.metrics import classification_report, accuracy_score
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        17
  versicolor       1.00      0.86      0.92        14
   virginica       0.88      1.00      0.93        14

    accuracy                           0.96        45
   macro avg       0.96      0.95      0.95        45
weighted avg       0.96      0.96      0.96        45
```

```
# Imprime a acurácia final do MLP para o Dataset Íris

accuracy = accuracy_score(y_test, y_pred)
accuracy_percentage = round(accuracy*100, 2)
print("Acurácia do MLP: " + str(accuracy_percentage) + "%")
```

```
Acurácia do MLP: 95.56%
```

Logo, a acurácia do MLP foi de 95.56%, um pouco menor do que o SVM testado acima.

Caso queira rodar o programa do MLP, segue o link do Kaggle professora:

https://www.kaggle.com/thiagohenriqueleite/mlp-iris