Criar um modelo para prever a chance de vitória de uma partida de futebol do time Avai FC, a partir de uma série de variáveis de entrada relacionadas com possíveis ações em decorrência da partida.

In [1]:
```
# Importa e inicializa findspark
import findspark
findspark.init()
```

In [2]:
```
# Imports
import pyspark
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.ml.feature import StringIndexer
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

In [3]:
```
# Spark Context
sc = SparkContext(appName = "ClassFut")
sc.setLogLevel("ERROR")
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/02/17 15:26:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
```

In [4]:
```
# Spark Session
spSession = SparkSession.builder.master("local").getOrCreate()
```

Carga de Dados

In [ ]:
```
```

Os dados serão carregados como RDD. Então removemos o cabeçalho.

In [5]:
```
# Carga dos dados saindo um RDD
dados_time_futebol = sc.textFile('dados/dataset2.csv')
```

In [6]:
```
dados_time_futebol.count()
```

Out[6]: 151

In [7]:
```
dados_time_futebol.take(5)
```

Out[7]:
```
['media_faltas_sofridas,media_faltas_recebidas,media_cartoes_recebidos,media_chutes_a_gol,resultado',
 '4.8,3,1.4,0.3,vitoria',
 '5.1,3.8,1.6,0.2,vitoria',
 '4.6,3.2,1.4,0.2,vitoria',
 '5.3,3.7,1.5,0.2,vitoria']
```

In [8]:
```
# Remove cabeçalho
dados_time_futebol_2 = dados_time_futebol.filter(lambda x: "media_faltas_sofridas" not in x)
dados_time_futebol_2.count()
```

Out[8]: 150

Limpeza e Transformação dos Dados

Aqui vamos separar as colunas por ",". Mapear e garantir o tipo de dado para um cálculo mais preciso. Converter o RDD em Data Frame. E então, criar, treinar e aplicar um indexador a variável Target.

In [9]:
```
# Separação das colunas
dados_time_futebol_3 = dados_time_futebol_2.map(lambda l: l.split(","))
```

In [10]:
```
# Mapeamento das colunas
dados_time_futebol_4 = dados_time_futebol_3.map(lambda p: Row(media_faltas_sofridas = float(p[0]),
                                                              media_faltas_recebidas = float(p[1]),
                                                              media_cartoes_recebidos = float(p[2]),
                                                              media_chutes_a_gol = float(p[3]),
                                                              resultado = p[4] ))
```

In [11]:
```
# Conv. RDD p DF
df_time = spSession.createDataFrame(dados_time_futebol_4)
df_time.cache()
```

Out[11]:
```
DataFrame[media_faltas_sofridas: double, media_faltas_recebidas: double, media_cartoes_recebidos: doubl
e, media_chutes_a_gol: double, resultado: string]
```

In [12]:
```
df_time.take(5)
```

Out[12]:
```
[Row(media_faltas_sofridas=4.8, media_faltas_recebidas=3.0, media_cartoes_recebidos=1.4, media_chutes_a_
gol=0.3, resultado='vitoria'),
 Row(media_faltas_sofridas=5.1, media_faltas_recebidas=3.8, media_cartoes_recebidos=1.6, media_chutes_a_
gol=0.2, resultado='vitoria'),
 Row(media_faltas_sofridas=4.6, media_faltas_recebidas=3.2, media_cartoes_recebidos=1.4, media_chutes_a_
gol=0.2, resultado='vitoria'),
 Row(media_faltas_sofridas=5.3, media_faltas_recebidas=3.7, media_cartoes_recebidos=1.5, media_chutes_a_
gol=0.2, resultado='vitoria'),
 Row(media_faltas_sofridas=5.1, media_faltas_recebidas=3.5, media_cartoes_recebidos=1.4, media_chutes_a_
gol=0.2, resultado='vitoria')]
```

In [13]:
```
# Indice numérico p a coluna de label target
stringIndexer = StringIndexer(inputCol = "resultado", outputCol = "idx_resultado")
```

In [14]:
```
# Fit string indexer
si_model = stringIndexer.fit(df_time)
```

In [15]:
```
# Aplica
df_time_final = si_model.transform(df_time)
```

In [16]:
```
df_time_final.select("resultado", "idx_resultado").distinct().collect()
```

Out[16]:
```
[Row(resultado='derrota', idx_resultado=0.0),
 Row(resultado='vitoria', idx_resultado=2.0),
 Row(resultado='empate', idx_resultado=1.0)]
```

Análise Exploratória

Verificaremos a correlação, a fim de buscar a força da relação entre a variável de saída com as variáveis de entrada, e entender melhor tal correlação dos dados, a fim de verificar quais variáveis possivelmente terão maior impacto nas previsões do modelo.

In [17]:
```
# Estatística descritiva
df_time_final.describe().show()
```

```
+-------+------------------+--------------------+--------------------+--------------------+-----------------+-------
--+-----------------+
|summary|media_faltas_sofridas|media_faltas_recebidas|media_cartoes_recebidos|media_chutes_a_gol|resulta
do|     idx_resultado|
+-------+------------------+--------------------+--------------------+--------------------+-----------------+-------
--+-----------------+
|  count|               150|                 150|                 150|                 150|      1
50|              150|
|   mean|   5.843333333333332|   3.0573333333333337|     3.758000000000001|1.1993333333333331|     NU
LL|              1.0|
| stddev|  0.8280661279778625|  0.43586628493669793|    1.7652982332594667|0.7622376689603465|     NU
LL|0.8192319205190404|
|    min|               4.3|                 2.0|                 1.0|                 0.1|  derro
ta|              0.0|
|    max|               7.9|                 4.4|                 6.9|                 2.5|  vitor
ia|              2.0|
+-------+------------------+--------------------+--------------------+--------------------+-----------------+-------
--+-----------------+
```

In [18]:
```
# Correlação da variável Target com as demais variáveis
for i in df_time_final.columns:
    if not(isinstance(df_time_final.select(i).take(1)[0][0], str)) :
        print("Correlação da variável idx_resultado com:", i, df_time_final.stat.corr('idx_resultado', i))
```

```
Correlação da variável idx_resultado com: media_faltas_sofridas -0.4600391565002369
Correlação da variável idx_resultado com: media_faltas_recebidas 0.6183715308237437
Correlação da variável idx_resultado com: media_cartoes_recebidos -0.649241830764174
Correlação da variável idx_resultado com: media_chutes_a_gol -0.5803770334306265
Correlação da variável idx_resultado com: idx_resultado 1.0
```

Pré-Processamento

In [ ]:
```
```

Resolvemos levar todas as variáveis preditoras a diante, então vamos criar e aplicar uma função para criar um RDD com o resultado e um vetor denso das variáveis preditoras para o modelo.

In [19]:
```
# Etiqueta o resultado com o vetor denso das variáveis preditoras
def transformaVar(row) :
    obj = (row["resultado"], row["idx_resultado"], Vectors.dense([row["media_faltas_sofridas"],
                                                                  row["media_faltas_recebidas"],
                                                                  row["media_cartoes_recebidos"],
                                                                  row["media_chutes_a_gol"]]))
    return obj
```

In [20]:
```
# Aplica
df_time_final_RDD = df_time_final.rdd.map(transformaVar)
```

In [21]:
```
df_time_final_RDD.take(5)
```

Out[21]:
```
[('vitoria', 2.0, DenseVector([4.8, 3.0, 1.4, 0.3])),
 ('vitoria', 2.0, DenseVector([5.1, 3.8, 1.6, 0.2])),
 ('vitoria', 2.0, DenseVector([4.6, 3.2, 1.4, 0.2])),
 ('vitoria', 2.0, DenseVector([5.3, 3.7, 1.5, 0.2])),
 ('vitoria', 2.0, DenseVector([5.1, 3.5, 1.4, 0.2]))]
```

In [22]:
```
# Converte o RDD para DataFrame
df_spark = spSession.createDataFrame(df_time_final_RDD,["resultado", "label", "features"])
```

In [23]:
```
df_spark.cache()
```

Out[23]:
```
DataFrame[resultado: string, label: double, features: vector]
```

In [24]:
```
df_spark.select("resultado", "label", "features").show(10)
```

```
[Stage 38:>                                                         (0 + 2) / 2]

+---------+-----+-----------------+
|resultado|label|         features|
+---------+-----+-----------------+
|  vitoria|  2.0|[4.8,3.0,1.4,0.3]|
|  vitoria|  2.0|[5.1,3.8,1.6,0.2]|
|  vitoria|  2.0|[4.6,3.2,1.4,0.2]|
|  vitoria|  2.0|[5.3,3.7,1.5,0.2]|
|  vitoria|  2.0|[5.1,3.5,1.4,0.2]|
|  vitoria|  2.0|[4.9,3.0,1.4,0.2]|
|  vitoria|  2.0|[4.7,3.2,1.3,0.2]|
|  vitoria|  2.0|[4.6,3.1,1.5,0.2]|
|  vitoria|  2.0|[5.0,3.6,1.4,0.2]|
|  vitoria|  2.0|[5.4,3.9,1.7,0.4]|
+---------+-----+-----------------+
only showing top 10 rows
```

Machine Learning

Será escolhido o algorítmo Decision Tree para criação deste modelo, devido as características apresentadas do problema de negócio.

Vamos dividir os dados para treinar em 75% e 25% para testar o modelo. Criaremos, treinaremos o modelo e calcularemos a acurácia e a confusion matrix.

In [25]:
```
# Dados de treino e de teste
(dados_treino, dados_teste) = df_spark.randomSplit([0.75, 0.25])
```

In [26]:
```
dados_treino.count()
```

Out[26]:
```
113
```

In [27]:
```
dados_teste.count()
```

Out[27]:
```
37
```

In [28]:
```
# Cria o objeto
dtClassifer = DecisionTreeClassifier(maxDepth = 2, labelCol = "label", featuresCol = "features")
```

In [29]:
```
# Treina objeto com dados para criar o modelo
modelo = dtClassifer.fit(dados_treino)
```

In [30]:
```
# Previsões com dados de teste
previsoes = modelo.transform(dados_teste)
```

In [31]:
```
previsoes
```

Out[31]:
```
DataFrame[resultado: string, label: double, features: vector, rawPrediction: vector, probability: vecto
r, prediction: double]
```

In [32]:
```
previsoes.select("resultado", "label", "prediction", "probability").collect()
```

Out[32]:
```
[Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=1.0, probability=DenseVector([0.0909, 0.9091, 0.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='vitoria', label=2.0, prediction=2.0, probability=DenseVector([0.0, 0.0, 1.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=0.0, probability=DenseVector([0.9688, 0.0312, 0.0])),
 Row(resultado='derrota', label=0.0, prediction=1.0, probability=DenseVector([0.0909, 0.9091, 0.0])),
 Row(resultado='empate', label=1.0, prediction=1.0, probability=DenseVector([0.0909, 0.9091, 0.0])),
 Row(resultado='empate', label=1.0, prediction=1.0, probability=DenseVector([0.0909, 0.9091, 0.0])),
 Row(resultado='empate', label=1.0, prediction=1.0, probability=DenseVector([0.0909, 0.9091, 0.0])),
 Row(resultado='empate', label=1.0, prediction=1.0, probability=DenseVector([0.0909, 0.9091, 0.0])),
 Row(resultado='empate', label=1.0, prediction=1.0, probability=DenseVector([0.0909, 0.9091, 0.0])),
 Row(resultado='empate', label=1.0, prediction=1.0, probability=DenseVector([0.0909, 0.9091, 0.0])),
 Row(resultado='empate', label=1.0, prediction=1.0, probability=DenseVector([0.0909, 0.9091, 0.0])),
 Row(resultado='empate', label=1.0, prediction=1.0, probability=DenseVector([0.0909, 0.9091, 0.0])),
 Row(resultado='empate', label=1.0, prediction=1.0, probability=DenseVector([0.0909, 0.9091, 0.0]))]
```

In [33]:
```
# Acurácia
avaliador = MulticlassClassificationEvaluator(predictionCol = "prediction",
                                              labelCol = "label",
                                              metricName = "accuracy")
```

In [34]:
```
avaliador.evaluate(previsoes)
```

Out[34]:
```
0.9459459459459459
```

In [35]:
```
# Confusion matrix
previsoes.groupBy("resultado", "label", "prediction").count().show()
```

```
+---------+-----+----------+-----+
|resultado|label|prediction|count|
+---------+-----+----------+-----+
|  vitoria|  2.0|       2.0|   13|
|  derrota|  0.0|       0.0|   13|
|  derrota|  0.0|       1.0|    2|
|   empate|  1.0|       1.0|    9|
+---------+-----+----------+-----+
```

Nosso modelo apresentou uma ótima performance nos testes. Apesar de performar acima de 94% de acerto, ficam minhas considerações para dedicarmos mais tempo a melhorias deste modelo, como testando outros hiperparâmetros para o Decision Tree, ou até mesmo testando os dados com outros algorítmos, a fim de possivelmente obter uma previsão mais acertiva.