Instituto Federal do Norte de Minas Gerais

IFNMG – Campus Montes Claros

Bacharelado em Ciência da Computação

Disciplina de Inteligência Artificial

Seminário:

Random Forest

David Walter Jansen (davidwalterjansen@gmail.com)

Iarah Gonçalves de Almeida (iarahgda@gmail.com)

Paulo Augusto Borges de Matos

Última atualização: 1 de junho de 2019

Sumário

Resumo

O presente trabalho é referente a um seminário sobre modelos de aprendizagem de máquina, proposto pela Professora Drª. Luciana Balieiro Cosme na disciplina de Inteligência Artificial do curso de Ciência da Computação - IFNMG (Campus Montes Claros).

O modelo escolhido pela nossa equipe foi o Random Forest (Floresta Aleatória). Esse é um modelo de aprendizado supervisionado usado tanto para regressão quanto classificação [1]. Nosso foco se dará sobre essa último. Falaremos sobre a aplicação do método, como são usadas as árvores de decisão (outro modelo de aprendizagem), bem como suas vantagens e desvantagens. Abordaremos também a seleção de características, uma das aplicações mais importantes desse modelo.

Capítulo 1

Random Forest

1.1 História e Motivação

O primeiro algoritmo para florestas de decisão aleatória foi proposto por Tim Kam Ho em 1995. Em seu artigo, Tim Kam Ho menciona que *as árvores derivadas de métodos tradicionais geralmente não podem ser expandidas até a complexidade arbitrária para possível perda de precisão de generalização em dados não vistos. A limitação da complexidade significaria precisão abaixo do ideal nos dados de treinamento* [3]. Esse fato serviu de motivação para o desenvolvimento de um novo método, baseado em princípios da modelagem estocástica[1], que utiliza a construção de várias árvores em subespaços selecionados aleatoriamente sobre espaço dos dados totais.

No trabalho inicial, Ho havia estabelecido as divisões em hiperplanos oblíquos. Posteriormente em 1998 [5], o mesmo autor mostrou que outros métodos de divisão são válidos. O algoritmo de Floresta Aleatória sofreu uma grande evolução com o trabalho de Breiman [2], onde foi introduzida a capacidade de medição da importância das características.

1.2 Ideia do Algoritmo

Suponha que você está montando sua grade para o próximo período e precisa escolher mais uma disciplina. Usando o paradigma da divisão e conquista, uma das possibilidades é pedir recomendações a seus colegas. Cada colega poderá lhe dar algumas recomendações. Com essas recomendações, você terá uma lista de matérias para completar sua grade. Agora, realize uma votação com base nas recomendações de seus colegas. A disciplina com mais votos será a sua escolha final.

[1] Um processo estocástico é formado por variáveis que se comportam, durante o tempo, de uma maneira onde pelo menos parte é considerada randômica.

O processo de escolha de disciplina que descrevemos é uma analogia ao funcionamento da decisão feita pela Floresta Aleatória. Chamamos de floresta a coleção de classificadores por árvores de decisão. Para ficar mais claro, iremos dividir a analogia feita em duas partes: a primeira parte é o recebimento de recomendações dos colegas. Nessa parte, o algoritmo da Floresta de decisão usa árvores de decisão. Cada colega (árvore) faz uma seleção das matérias que ele tem conhecimento. A segunda parte é o procedimento de votação. Após coletar todas as recomendações dos colegas, a votação para encontrar a melhor sugestão é uma dos processos do algoritmo de Florestas Aleatórias.

Na classificação, cada árvore vota e a classe mais popular é escolhida. Para a regressão, a média de todas as saídas de árvore é considerada como resultado final.

1.3 Funcionamento do Algoritmo

O algoritmo constrói o conjunto de árvores de decisão usando divisões aleatórias e independentes sobre o conjunto de dados. No processo de geração é usado um indicador de seleção de atributos, como ganho de informação, taxa de ganho e índice de Gini para cada atributo [1].

O funcionamento da Random Forest pode ser resumido em 4 etapas:

(1) Selecione amostras aleatórias de um determinado conjunto de dados.
(2) Construa uma árvore de decisão para cada amostra e obtenha um resultado de previsão de cada árvore de decisão.
(3) Faça um voto para cada resultado previsto.
(4) Selecione o resultado da previsão com o maior número de votos como previsão final.

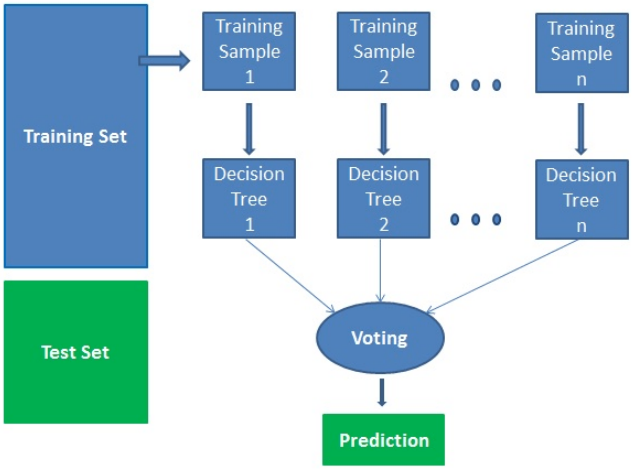


Figura 1.1: Divisão e conquista no algoritmo Random Forest[1]

1.4 Vantagens e Desvantanges

Vantagens

- O uso de muitas árvores de decisão torna a Random Forest uma método preciso e robusto.
- Como podemos fazer uso de várias árvores de decisão, esse método não sofre com *overfitting* [2].
- Como dissemos antes, podemos usar a Random Forest para classificação e regressão.
- É possível obter a importância relativa da característica, o que ajuda na seleção das melhores para treinar e testar o modelo.

Desvantagens

- Por usar as n-árvores de decisão sempre que se deseja fazer uma previsão e depois realizar a votação, a execução completa do algoritmo pode se tornar lenta.
- Esse modelo requer uma análise mais cuidadosa na interpretação em comparação com uma única árvore de decisão, onde apenas precisamos olhar para o caminho dessa árvore.
- A ordenação das características mais importantes pode alterar até para bases pequenas como a Iris. Isso ocorre devido à natureza aleatória do algoritmo na construção das árvores de decisão.

1.5 Coeficiente de Gini

Nos casos em que o conjunto de dados possui m atributos, decidir qual atributo deve ser escolhido para a raiz da árvore ou para um nó intermediário é um passo complicado e, muitas vezes, selecionar qualquer nó para a raiz pode resultar em resultados ruins e baixa acurácia, o que não ajudaria na construção de um modelo adequado.

Para tanto, alguns critérios podem ser utilizados, sendo o coeficiente de Gini (G) um deles. Este coeficiente é uma das métricas utilizadas para medir a frequência que um elemento escolhido aleatoriamente é identificado incorretamente. Isso significa que um atributo com um menor coeficiente deve ser preferido. O coeficiente é uma medida de desigualdade dado por um número entre 0 e 1, em que 0 significa completa igualdade e 1 corresponde à completa desigualdade.

[2] Overfitting é um fenômeno em que um modelo de aprendizado de máquina modela os dados de treinamento muito bem, mas não consegue ter um bom desempenho nos dados de teste.

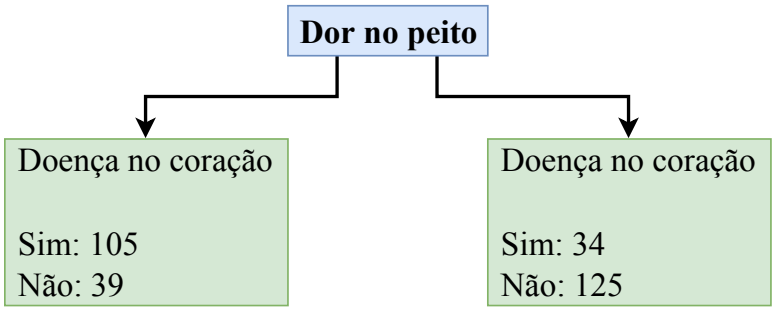


Figura 1.2: Nó de uma árvore de decisão.

Suponha que tenhamos o seguinte nó da Figura 1.2 cuja característica é dor no peito e a classificação realizada pela árvore de decisão é se o paciente tem ou não doença no coração. Para a folha da esquerda, o coeficiente de Gini é dado por:

$$\begin{aligned} G_{esq} &= 1 - (\text{probabilidade de sim})^2 - (\text{probabilidade de não})^2 \\ &= 1 - (\frac{105}{105+39})^2 - (\frac{39}{105+39})^2 \\ &= 0.395 \end{aligned}$$

Portanto, a impureza da folha esquerda é de 0.395. Faremos o mesmo processo para a folha da direita:

$$\begin{aligned} G_{dir} &= 1 - (\text{probabilidade de sim})^2 - (\text{probabilidade de não})^2 \\ &= 1 - (\frac{34}{34+125})^2 - (\frac{125}{34+125})^2 \\ &= 0.336 \end{aligned}$$

Após o cálculo do coeficiente de Gini para as duas folhas podemos calcular a impureza do nó intermediário (ou raiz). Como o nó da esquerda representa 144 pacientes e o da direita 159 pacientes, o coeficiente de Gini para o nó pai representa a média ponderada das impurezas das folhas. Para tanto, seja x = número de pacientes da folha esquerda = 144 e y = número de pacientes da folha direita = 159, G_{raiz} é dado por:

$$\begin{aligned} G_{raiz} &= \frac{x}{x+y} \cdot G_{esq} + \frac{y}{x+y} \cdot G_{dir} \\ &= \frac{144}{144+159} \cdot 0.395 + \frac{159}{144+159} \cdot 0.336 \\ &= 0.364 \end{aligned}$$

O cálculo do coeficiente de Gini é utilizado para definir qual nó terá prioridade para ser a raiz, em que o de menor impureza é escolhido. Dessa forma, os 144 pacientes do nó esquerdo serão

classificados de acordo com o lado esquerdo da árvore de decisão, enquanto os 159 pacientes do nó direito serão classificados conforme o lado direito da mesma árvore.

O procedimento de criação de nós é feito com as demais características do conjunto de dados até que o coeficiente de Gini da próxima característica não seja menor que o coeficiente da característica anterior.

1.6 Pseudocódigos

O algoritmo é dividido em duas partes: (1) criação, em que o modelo é construído e; (2) predição, que utiliza o modelo treinado para realizar as predições. O pseudocódigo dessas duas fases será apresentado a seguir.

Pseudocódigo 1 Criação da floresta aleatória

1: selecione, de forma aleatória, k características dentre as m características totais ($k < m$)
2: dentre as k características, calcule o nó d utilizando o coeficiente de Gini
3: divida o nó em nós filhos utilizando o coeficiente de Gini
4: repita os passos 1 ao 3 até que l nós tenham sido alcançados
5: construa a floresta repetindo os passos 1 ao 4 por n vezes e assim crie n árvores

Na criação, primeiramente selecionamos k características de um total de m características de forma aleatória. Após, utilizamos essas k características para encontrar a raiz a partir de uma abordagem de melhor divisão, nesse caso utilizamos o coeficiente de Gini. Então, calculamos os nós filhos usando a mesma abordagem de divisão, isto é, o coeficiente de Gini. Os 3 passos iniciais são repetidos até que tenhamos uma árvore com uma raiz e os possíveis resultados da classificação como nós folha. Finalmente, repetimos os passos 1 ao 4 para criar n árvores aleatórias. Essas árvores aleatórias constituem a floresta aleatória.

Pseudocódigo 2 Predição com a floresta aleatória

1: dada as características da instância de teste, as utiliza em cada árvore de decisão criada na etapa anterior para predizer o resultado, guardando o resultado predito (classificação)
2: calcula os votos de cada classificação predita
3: considera a classificação mais votada como a predição final do algoritmo de floresta aleatória

Para a fase de predição utilizando a floresta aleatória treinada, as características da instância de teste passam por todas as árvores de decisão criadas, isto é, pela floresta aleatória. O resultado de cada árvore é guardado e o resultado final é dado, comumente, pelo conceito de maioria simples.

Capítulo 2

Exemplo de Aplicação

A *SciKit Learn* disponibiliza uma implementação do algoritmo RandomForestClassifier. Essa implementação foi utilizada em um trabalho realizado em 2018, em que utilizamos uma base de dados de prognóstico de câncer de mama. (WPBC) [4].

Tópicos em IC: Análise exploratória de dados.

Equipe: David Jansen, Iarah Gonçalves de Almeida, Paulo Borges

1) Introdução

A análise de dados será feita sobre o Breast Cancer Wisconsin (Prognostic) Data Set. (http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Prognostic%29) A manipulação dos dados para análise será feita utilizando a linguagem Python no ambiente Jupyter.

2) Informação dos Atributos

1) ID number

2) Outcome (R = recur, N = nonrecur)

3) Time (recurrence time if field 2 = R, disease-free time if field 2 = N)

4-33) Ten real-valued features are computed for each cell nucleus:

1. radius (mean of distances from center to points on the perimeter)
2. texture (standard deviation of gray-scale values)
3. perimeter
4. area
5. smoothness (local variation in radius lengths)
6. compactness (perimeter^2 / area - 1.0)
7. concavity (severity of concave portions of the contour)
8. concave points (number of concave portions of the contour)
9. symmetry
10. fractal dimension ("coastline approximation" - 1)

Several of the papers listed above contain detailed descriptions of how these features are computed.

The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 4 is Mean Radius, field 14 is Radius SE, field 24 is Worst Radius.

Values for features 4-33 are recoded with four significant digits.

34) Tumor size - diameter of the excised tumor in centimeters

35) Lymph node status - number of positive axillary lymph nodes observed at time of surgery.

3) Preparação do Ambiente Jupyter

In [29]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from urllib.request import urlopen

#URL onde se encontra a base de dados.
UCI_data_URL = 'https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer
-wisconsin/wpbc.data'

#Nomeação das colunas.
names = ['id_number', 'outcome', 'time', 'radius_mean',
         'texture_mean', 'perimeter_mean', 'area_mean',
         'smoothness_mean', 'compactness_mean', 'concavity_mean',
         'concave_points_mean', 'symmetry_mean',
         'fractal_dimension_mean', 'radius_se', 'texture_se',
         'perimeter_se', 'area_se', 'smoothness_se',
         'compactness_se', 'concavity_se', 'concave_points_se',
         'symmetry_se', 'fractal_dimension_se',
         'radius_worst', 'texture_worst', 'perimeter_worst',
         'area_worst', 'smoothness_worst',
         'compactness_worst', 'concavity_worst',
         'concave_points_worst', 'symmetry_worst',
         'fractal_dimension_worst', 'tumor_size', 'lymph_node_status']

#Leitura do arquivo para o formato csv.
wpbc = pd.read_csv(urlopen(UCI_data_URL), names=names)

#Mostrando resultado em forma de tabela (10 primeiros).
wpbc.head(10)
```

Out[29]:

	id_number	outcome	time	radius_mean	texture_mean	perimeter_mean	area_me
0	119513	N	31	18.02	27.60	117.50	1013.0
1	8423	N	61	17.99	10.38	122.80	1001.0
2	842517	N	116	21.37	17.44	137.50	1373.0
3	843483	N	123	11.42	20.38	77.58	386.1
4	843584	R	27	20.29	14.34	135.10	1297.0
5	843786	R	77	12.75	15.29	84.60	502.7
6	844359	N	60	18.98	19.61	124.40	1112.0
7	844582	R	77	13.71	20.83	90.20	577.9
8	844981	N	119	13.00	21.82	87.50	519.8
9	845010	N	76	12.46	24.04	83.97	475.9

10 rows × 35 columns

4) Tratamento do Conjunto de Dados para análise

O atributo 'outcome' está representado com um objeto string. Faremos um mapeamento do atributo para valores numéricos, onde N = 0.0 e R = 1.0. Isso ajuda na plotagem de gráficos e na aplicação de alguns modelos.

Algumas informações podem ser obtidadas através do arquivo wpbc.names (https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wpbc.names):

- O atributo 'lymph_node_status' está ausente em 4 amostras.
- 151 amostras não apresentam recorrência('outcome' = 0), 47 apresentam recorrência('outcome' = 1).

Dada a primeira informação na lista acima, iremos retirar essas 4 amostras de nossa análise.

As entradas da tabela serão passadas para o formato ***float64***.

As colunas ***id_number, time*** serão removidas do nosso conjunto de dados por não se tratarem de informações relevantes em nossa análise.

In [30]:

```
# Mapeia a coluna 'outcome' para valores numéricos, 'R' de recur será 1, 'N' de nonrecu
r será 0
wpbc['outcome'] = wpbc['outcome'].map({'R':1.0, 'N':0.0})

# Retira os 4 valores null da coluna 'lymph_node_status'
wpbc = wpbc[wpbc['lymph_node_status'] != '?']

# Converte todo o dataframe para float64
wpbc = wpbc.astype('float64')

#Removendo colunas "id_number" e "time"
wpbc = wpbc.drop(columns=['id_number', 'time'])

# Reseta os indices para não ter falha dos valores retirados
wpbc.reset_index(inplace=False)

wpbc.head(10)
```

Out[30]:

	outcome	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
0	0.0	18.02	27.60	117.50	1013.0	0.09489
1	0.0	17.99	10.38	122.80	1001.0	0.11840
2	0.0	21.37	17.44	137.50	1373.0	0.08836
3	0.0	11.42	20.38	77.58	386.1	0.14250
4	1.0	20.29	14.34	135.10	1297.0	0.10030
5	1.0	12.75	15.29	84.60	502.7	0.11890
7	1.0	13.71	20.83	90.20	577.9	0.11890
8	0.0	13.00	21.82	87.50	519.8	0.12730
9	0.0	12.46	24.04	83.97	475.9	0.11860
10	0.0	16.02	23.24	102.70	797.8	0.08206

10 rows × 33 columns

5) Informações da tabela

Podemos ter mais algumas informações estatísticas do conjunto de dados, usando a função ***describe***.
Informações dos tipos de dados, podem ser obtidos com a função ***info***.

In [31]:

```
wpbc.describe()
```

Out[31]:

	outcome	radius_mean	texture_mean	perimeter_mean	area_mean	smoot
count	194.000000	194.000000	194.000000	194.000000	194.000000	194.00
mean	0.237113	17.402320	22.300979	114.781495	969.092268	0.1027
std	0.426413	3.171672	4.335292	21.430694	353.159959	0.0126
min	0.000000	10.950000	10.380000	71.900000	361.600000	0.0749
25%	0.000000	15.052500	19.342500	98.160000	702.525000	0.0939
50%	0.000000	17.290000	21.795000	113.700000	929.100000	0.1022
75%	0.000000	19.580000	24.782500	129.650000	1193.500000	0.1113
max	1.000000	27.220000	39.280000	182.100000	2250.000000	0.1447

8 rows × 33 columns

In [32]:

```
wpbc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 194 entries, 0 to 197
Data columns (total 33 columns):
outcome                    194 non-null float64
radius_mean                194 non-null float64
texture_mean               194 non-null float64
perimeter_mean             194 non-null float64
area_mean                  194 non-null float64
smoothness_mean            194 non-null float64
compactness_mean           194 non-null float64
concavity_mean             194 non-null float64
concave_points_mean        194 non-null float64
symmetry_mean              194 non-null float64
fractal_dimension_mean     194 non-null float64
radius_se                  194 non-null float64
texture_se                 194 non-null float64
perimeter_se               194 non-null float64
area_se                    194 non-null float64
smoothness_se              194 non-null float64
compactness_se             194 non-null float64
concavity_se               194 non-null float64
concave_points_se          194 non-null float64
symmetry_se                194 non-null float64
fractal_dimension_se       194 non-null float64
radius_worst               194 non-null float64
texture_worst              194 non-null float64
perimeter_worst            194 non-null float64
area_worst                 194 non-null float64
smoothness_worst           194 non-null float64
compactness_worst          194 non-null float64
concavity_worst            194 non-null float64
concave_points_worst       194 non-null float64
symmetry_worst             194 non-null float64
fractal_dimension_worst    194 non-null float64
tumor_size                 194 non-null float64
lymph_node_status          194 non-null float64
dtypes: float64(33)
memory usage: 51.5 KB
```

6) Exploração de Dados

Em nosso trabalho intermediário (https://github.com/DWalterJansen/topicosIC/blob/master/Equipe/Trabalho%20Intermedi%C3%A1rio/trabalho_ analisamos apenas as caracteristicas de valor médio(_mean), tumor_size, e lymph_node_status para avaliar a influência na recorrência ou não recorrência do cancêr de mama. Os resultados lá obtidos, nos motivaram à apronfunar na análise. Nessa versão final do trabalho para a disciplina, iremos aumentar o número de características observadas e aplica-lás em alguns modelos de classificação.

6.1) Floresta Aleatória com conjunto de treinamento e teste

Iniciamos usando uma ***RandomForestClassifier*** aplicada ao nosso dataframe ***wpbc***, onde usaremos as 32 caracterísicas para fazer a predição do valor na coluna ***outcome***.

6.1.1) Procedimentos para classificação

Primeiramente, vamos separar nossas colunas em dependentes e independentes. A divisão entre o conjunto de treinamento e teste será feito usando o algoritmo **K-Fold** que realiza o *split* no conjunto de dados.

6.1.2) Treinamento do Modelo e Resultado do treinamento

Depois de separarmos os conjuntos, podemos aplicar o treinamento. Em seguida, verificamos a precisão do modelo atráves da média dos resultados obtidos para cada *split*, o que nos diz quão frequente nosso classificador estará correto.

Durante a execução do algoritmo, guardamos sempre o melhor resultado de predição. Usaremos esse resultado para medir a importância das características para a predição.

6.1.3) Importância das Variáveis:

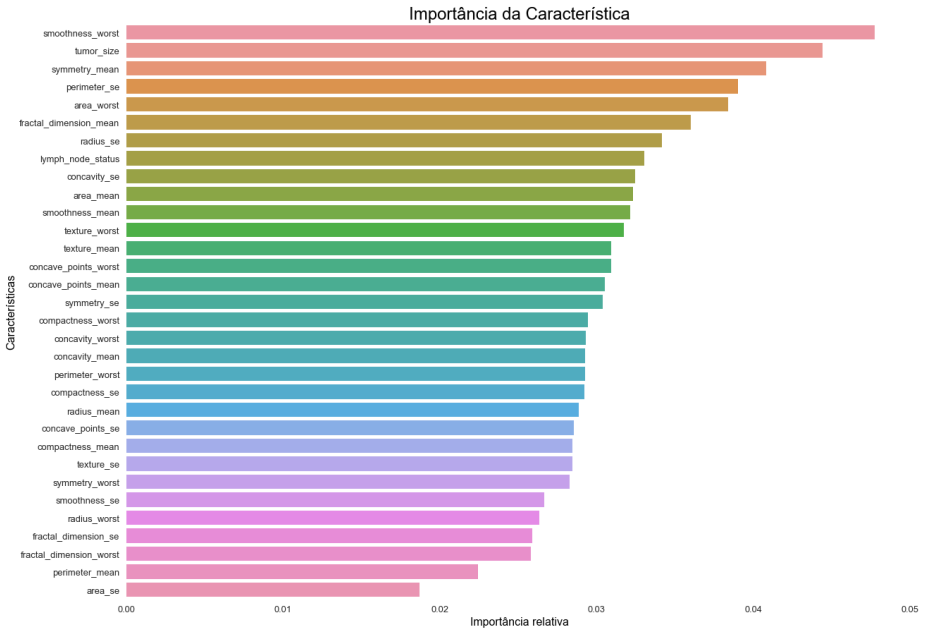
Uma outra importante funcionalidade da ***RandomForestClassifier*** é a possibilidade de exibirmos a importância das características para o modelo.

In [33]:

```
################################################################################
#6.1)

#Separa as colunas em dependentes e independentes

namesrfc = ['radius_mean',
         'texture_mean', 'perimeter_mean', 'area_mean',
         'smoothness_mean', 'compactness_mean', 'concavity_mean',
         'concave_points_mean', 'symmetry_mean',
         'fractal_dimension_mean', 'radius_se', 'texture_se',
         'perimeter_se', 'area_se', 'smoothness_se',
         'compactness_se', 'concavity_se', 'concave_points_se',
         'symmetry_se', 'fractal_dimension_se',
         'radius_worst', 'texture_worst', 'perimeter_worst',
         'area_worst', 'smoothness_worst',
         'compactness_worst', 'concavity_worst',
         'concave_points_worst', 'symmetry_worst',
         'fractal_dimension_worst', 'tumor_size', 'lymph_node_status']

X = wpbc[namesrfc]  # Características
y = wpbc['outcome']  # Rótulo


# Treinamento do modelo
# Importa 'RandomForestClassifier' de 'sklearn.ensemble'
from sklearn.ensemble import RandomForestClassifier
# Importa 'metrics' de 'sklearn'
from sklearn import metrics

# Cria um classificador gaussiano
clf = RandomForestClassifier(n_estimators=100)

# Importa o K-fold para separar entre treino e teste
from sklearn.model_selection import KFold

# Valor do K (número de divisões na base de dados)
kf = KFold(n_splits = 10)

best_precision = 0.0000
sum_precisions = 0.0000

################################################################################
# 6.1.1)
i = 0
for train_index, test_index in kf.split(X):
    i = i + 1 #contado de split
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Treina o modelo usando o conjunto de treinamento
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    # Modelo de precisão, que diz quão frequente o classificador está correto
    precision = metrics.accuracy_score(y_test, y_pred)
    print("Precisão encontrada para %dº split: " %(i) ,end="")
    print(precision)
    sum_precisions = sum_precisions + precision
    if best_precision < precision:
```

```
        best_precision = precision
        clf_best = clf
        y_test_best = y_test
        y_pred_best = y_pred

print("Resultados Atuais..")
print("\nPrecisão média para as %d predições: " %i, end="")
print(sum_precisions/i);
print("Melhor resultado: ", end="")
print(best_precision)

################################################################################
# 6.1.3

# Ordenação das as características importantes pela pontuação usando a iterção com a me
lhor precisão
print("\nCaracterísticas ordenadas pela pontuação:")
# O número dentro de round é o limite de casas decimais
result = sorted(zip(map(lambda x: round(x, 6), clf_best.feature_importances_), namesrfc
), reverse=True)

# Criando dataframe com o resultado
df = pd.DataFrame(data=result)
df = df.rename(index=str, columns={0: "importance", 1: "feature"})

import matplotlib.pyplot as plt
#Configurações para exibição do gráfico
sns.set_color_codes("dark")
sns.set_style("white")
sns.set_context("talk")
plt.figure(figsize=(20,15))
g = sns.barplot(x="importance", y="feature", data=df)
g.axes.set_title('Importância da Característica', fontsize=24,color="black",alpha=2)
g.set_xlabel("Importância relativa", size = 16,color="black")
g.set_ylabel("Características", size = 16,color="black")
sns.despine(left=True, bottom=True)
```

```
Precisão encontrada para 1º split: 0.7
Precisão encontrada para 2º split: 0.8
Precisão encontrada para 3º split: 0.65
Precisão encontrada para 4º split: 0.75
Precisão encontrada para 5º split: 0.9473684210526315
Precisão encontrada para 6º split: 0.631578947368421
Precisão encontrada para 7º split: 0.631578947368421
Precisão encontrada para 8º split: 0.631578947368421
Precisão encontrada para 9º split: 0.7894736842105263
Precisão encontrada para 10º split: 1.0
Resultados Atuais..

Precisão média para as 10 predições: 0.7531578947368421
Melhor resultado: 1.0

Características ordenadas pela pontuação:
```



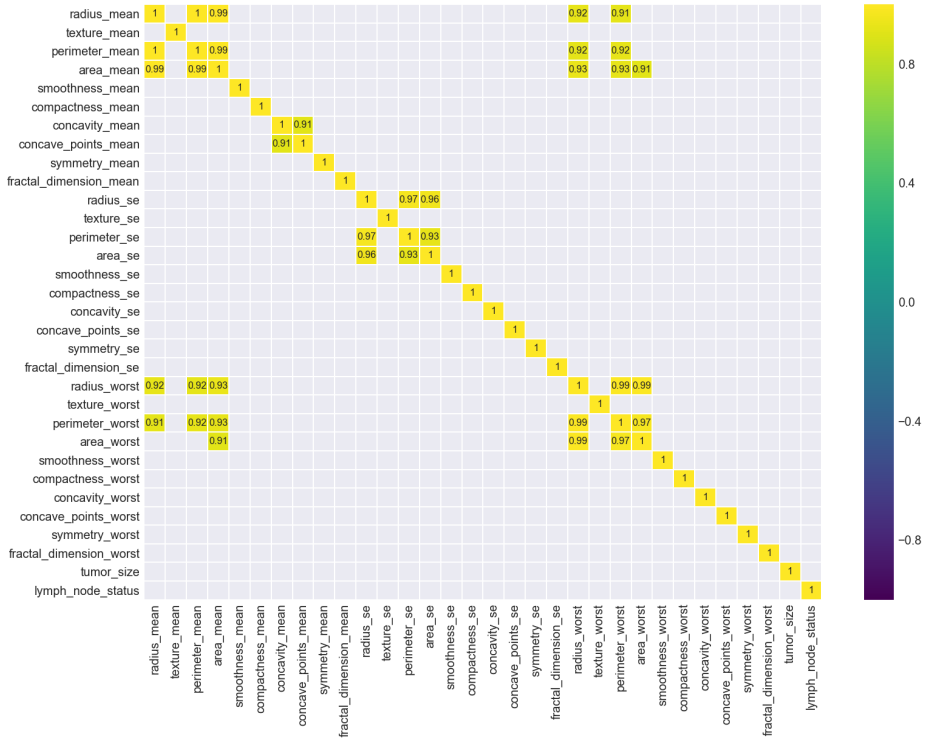
6.2) Correlação

Nossa base de dados possui muitas características, portando iremos tentar reduzir essa quantidade olhando para os atributos correlacionados.

Uma ferramenta que nos permite aprofundar um pouco mais na exploração da correlação de características é mapa de calor:

In [34]:

```
# Gerando a correlação das características
corr = wpbc.drop('outcome', axis=1).corr() # We already examined SalePrice correlations
plt.figure(figsize=(25, 18))
sns.set_context("poster")
sns.set(font_scale=2.0)


# Exibindo as correlações maiores que 0.75 e menor que -0.75
c = sns.heatmap(corr[(corr >= 0.9) | (corr <= -0.9)],
            cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=1,
            annot=True, annot_kws={"size": 16}, square=False);

size = len(namesrfc)
coor_features = list();
data = (corr[(corr >= 0.9) | (corr <= -0.9)])
for i in range(0,size):
    for j in range(i):
        if (data.iloc[i,j] > 0):
            coor_features.append([namesrfc[i], namesrfc[j]])
```



Referências Bibliográficas

[1] Understanding random forests classifiers in python. https://www.datacamp.com/community/tutorials/random-forests-classifier-python#algorithm. Accessado: 2019-05-25.

[2] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[3] Tin Kam Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, pages 278–, Washington, DC, USA, 1995. IEEE Computer Society.

[4] David Walter Jansen, Paulo Augusto Borges de Matos, and Iarah Gonçalves de Almeida. *Random Forest para predição sobre a Breast Cancer Wisconsin (Prognostic) Data Set.* 2018.

[5] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, Aug 1998.