Projeto de Data Science

Medicina Personalizada

Redefinindo o Tratamento de Câncer

Fellipe Augusto Soares Silva

2020

Sumário

Resumo 5

1 – Introdução 6

2 – Problema de Negócio 7

3 – Datasets utilizados 8

3.1 training_variants / test_variants 8

3.2 training_text / test_text 9

4 – Dicionário de Dados 10

5 – Bibliotecas utilizadas 11

6 – Feature Engineering 12

7 – Análise Quantitativa 13

7.1 Palavras mais frequentes encontradas nas evidências clínicas 13

7.2 Palavras menos frequentes encontradas nas evidências clínicas 14

7.3 Wordcloud de palavras com frequência maior de 300 15

7.4 Procurando por associações entre palavras 16

1. Cancer 16

2. Breast 17

3. Cbl (Casitas B-lineage Lymphoma) 18

7.5 Palavras que aparecem juntas e suas interligações 18

7.6 Distribuição de Letras 20

8 – Machine Learning 21

8.1 Troubleshotting 21

8.2 Distribuição de Classes 22

8.3 Neural Network 22

8.3.1. Confusion Matrix e Acurácia 24

8.3.2. Palavras mais importantes de acordo com o modelo preditivo 26

8.4 Decision Tree 27

8.4.1. Confusion Matrix e Acurácia 28

8.4.2. Palavras mais importantes de acordo com o modelo preditivo 29

8.5 K – Nearest Neighbors 30

8.5.1. Confusion Matrix e Acurácia 31

8.5.2. Palavras mais importantes de acordo com o modelo preditivo 33

8.6 Random Forest 33
8.6.1. Confusion Matrix e Acurácia 37
8.6.2. Palavras mais importantes de acordo com o modelo preditivo 38
8.7 Extreme Gradient Boosting (XGBoost) 39
8.7.1. Confusion Matrix e Acurácia 41
8.7.2. Palavras mais importantes de acordo com o modelo preditivo 42
9 – Conclusões 43
9.1 Apuração Modelos Preditivos 43
9.2 Consolidação das Palavras Mais Importantes 44
9.3 Consolidação das Palavras Menos Importantes 44
9.4 Letras por Palavras Mais Relevantes 45
9.5 Letras por Palavras Menos Relevantes 45
9.6 Considerações Finais 46
Código Fonte 47

Lista de Figuras

Figura 1 - Dataset training_variants / test_variants 8
Figura 2 - Dataset training_text / test_text 9
Figura 3 - Dicionário de Dados 10
Figura 4 - Palavras mais frequentes com Lemmatization 13
Figura 5 - Palavras mais frequentes com Stemming 14
Figura 6 - Palavras menos frequentes com Lemmatization 14
Figura 7 - Wordcloud 15
Figura 8 - Associações com "cancer" 16
Figura 9 - Associações com "breast" 17
Figura 10 - Associações com "cbl" 18
Figura 11 - Interligações entre as palavras 19
Figura 12 - Letras x Palavras 20
Figura 13 - Distribuição de Classes 21
Figura 14 - Distribuição de Classes 22
Figura 15 - Rede Neural Artificial 23
Figura 16 - Confusion Matrix Neural Network 24
Figura 17 - Palavras Mais Importantes (Neural Network) 26
Figura 18 - Decision Tree 27
Figura 19 - Decision Tree e Erro Relativo 28
Figura 20 - Confusion Matrix Decision Tree 28
Figura 21 - K - Nearest Neighbors Model 30
Figura 22 - Confusion Matrix (K - Nearest Neighbors) 31
Figura 23 - Palavras Mais Importantes (KNN) 33
Figura 24 - Random Forest 34
Figura 25 - Underfitting x Overfitting 35
Figura 26 - Underfitting x Ideal x Overfitting 36
Figura 27 - Curva de Acurácia do modelo RandomForest 36
Figura 28 - Confusion Matrix Random Forest 37
Figura 29 - Palavras Mais Importantes (KNN) 38
Figura 30 - XGBoost Tree 40
Figura 31 - Confusion Matrix XGBoost 41
Figura 32 - Modelos x Acurácia 43
Figura 33 - Palavras Mais Importantes (apuradas pelos modelos preditivos) 44
Figura 34 - Palavras Menos Importantes (apuradas pelos modelos preditivos) 44
Figura 35 - Letras por Palavras Mais Relevantes (apuradas pelos modelos preditivos) 45
Figura 36 - Letras por Palavras Menos Relevantes (apuradas pelos modelos preditivos) 45

Resumo

O presente projeto foi desenvolvido com a finalidade de implementar medicina personalizada na identificação e tratamento de mutações cancerígenas em seres humanos. Contamos com uma base de conhecimento anotada por especialistas patológicos reunidas em um dataset fornecido pela MSKCC, Memorial Sloan Kettering Cancer Center.

Para esse projeto foi preciso desenvolvê – lo em duas etapas onde a finalidade da primeira etapa foi analisar detalhadamente as evidências clínicas, onde tínhamos as palavras mais frequentes, quais as correlações entre termos, como se comportavam as interligações entre frases, dentre outras análises que resultaram em insights valiosos ao projeto.

Já a segunda etapa concentrou esforços na construção de modelos preditivos de modo que fosse possível prever à qual classe novas e desconhecidas evidências clínicas se encaixavam após modelos treinados serem testados. Esta etapa também agregou valor ao projeto pois foi possível explorar toda a capacidade dos modelos de Machine Learning, não limitando apenas à Acurácia e Confusion Matrix para análise de métricas estatísticas, mas também adquirindo dos modelos as palavras mais relevantes para prever cada classe de mutação que desencadeia o câncer em questão.

Um projeto desafiador e nada trivial primeiro por estamos lidando com riscos inerentes à saúde de seres humanos e segundo por estarmos analisando mais de 40.000 informações por evidência anotada pelos especialistas. Temos aproximadamente 5 milhões de palavras ao todo.

Assim, neste projeto é possível encontrar processamento de linguagem natural com pacotes da linguagem de programação R, pré – processamento de textos, frases, palavras e termos mais relevantes, análise quantitativa com diversos gráficos exemplificando cada etapa do processo de análise, escolha da melhor esparsidade dentro de uma matriz termo – documento, construção e análises de métricas de modelos de Machine Learning como Redes Neurais Artificiais, Árvore de Decisão, K – Nearest Neighbors, Random Forest e Extreme Gradient Boosting (XGBoost).

Palavras – Chave: Data Science, Medicina Personalizada, Diagnosticar Câncer, Mutações Genéticas, Machine Learning, Processamento de Linguagem Natural, NLP, Análises Quantitativas, Redes Neurais Artificiais, Decision Tree, KNN, RandomForest, XGBoost

1 – Introdução

Este projeto foi desenvolvido em linguagem de programação R utilizando o RStudio[1] como plataforma de desenvolvimento e teste do script[2].

Foram utilizadas também algumas bibliotecas com pacotes essenciais para o desenvolvimento do código, como por exemplo o ggplot para criação de gráficos, o caret para o aprendizado de máquina, o dplyr para manipulação de dados, dentre outros que serão explicados nos capítulos a seguir.

Como todo projeto de Data Science, antes de iniciar a parametrização do modelo preditivo é preciso realizar a análise exploratória e conhecer o conjunto de dados, ou *dataset*, porém este não é um projeto trivial, visto que temos um dataset com transcrição de textos e não apenas variáveis numéricas e categóricas. Certamente um projeto desafiador, que consequentemente trouxe oportunidades.

Durante a leitura do dataset houve uma limitação na linguagem R com os separadores de colunas dos dados pois temos 2 separadores e o R aceita apenas um tipo. Outro problema está na quantidade de classes que temos para prever, algumas apresentam limitações em nosso projeto. Isso não demonstrou ser impeditivo na criação do modelo de aprendizagem de máquina, porém é possível realizar otimizações na coleta de dados de modo que a confiabilidade seja maior no final do processo.

Este é um grande projeto e por isso será dividido em duas partes, a primeira focando no tratamento de textos, estudo das palavras que mais aparecem, eliminação das palavras menos frequentes construindo assim um dataset mais ajustado para a segunda parte que tem foco em modelos de Machine Learning, a saber: Neural Network (Redes Neurais), Decision Tree (Árvore de Decisão), K-Nearest Neighbors, Random Forest e XGBoost.

Serão apresentados todos os itens do script, comentados linha a linha e apresentando análises gráficas durante as fases do projeto para complementar o entendimento do desenvolvimento deste projeto de Data Science.

O início da solução de qualquer problema de negócio está no entendimento do problema propriamente dito o qual será apresentado no capítulo a seguir.

[1] Rstudio é um software livre de ambiente de desenvolvimento integrado para R, uma linguagem de programação para gráficos e cálculos estatísticos.

[2] Script é um conjunto de instruções em código, ou seja, escritas em linguagem de computador para que o mesmo execute diversas funções no interior de um programa.

2 – Problema de Negócio

Um tumor cancerígeno pode ter uma grande quantidade de mutações genéticas uma vez que foi sequenciado e distinguir essas mutações é um grande desafio. Atualmente este trabalho é feito manualmente por especialistas que documentam cada mutação associando o tumor à mutação correspondente.

O objetivo deste projeto é estudar os termos documentados por patologistas clínicos e utilizar text mining[3] para automatizar e maximizar o potencial de descoberta de tumores cancerígenos criando uma espécie de Medicina Personalizada através do potencial máximo fornecido pela análise de dados e processos de previsão com algoritmos de Machine Learning.

Muito tem sido dito no âmbito acadêmico e tecnológico sobre a Medicina de Precisão pois a associação de ambos pode fornecer ao paciente um tratamento mais preciso e eficiente, oferecendo assim uma abordagem individualizada de acordo com informações genéticas, biológicas e naturais de cada paciente.

Sistemas inteligentes podem auxiliar em diversos processos relacionados à área da saúde fazendo com que erros comuns sejam minimizados ou que até deixem de acontecer. Para isso é preciso que integrações de dados aconteçam, ou seja, diferentes fontes de informação sejam tratadas e unidas para um melhor entendimento de patologias.

Neste projeto utilizaremos datasets fornecido pelo Memorial Sloan Kettering Cancer Center (MSKCC[4]) que documentou a base de conhecimento anotada manualmente por especialistas, pesquisadores e oncologistas de nível mundial. Com essa base de conhecimento vamos criar cinco modelos diferentes de Machine Learning para classificar automaticamente as variações genéticas.

Interpretar evidências clínicas é desafiador para humanos e muito mais desafiador para máquinas, portanto modelar as evidências fornecidas pelo MSKCC será crítico para o sucesso do projeto. Assim, para começar o projeto vamos estudar a base de conhecimento fornecida e como cada variável está associada à mutação genética.

[3] Text Mining: conhecida também como mineração de dados textuais e semelhante à análise textual, refere-se ao processo de obtenção de informações importantes de um texto
[4] https://www.mskcc.org/

3 – Datasets utilizados

Para esse problema de negócio a “Memorial Sloan Kettering Cancer Center” forneceu os seguintes conjuntos de dados:

- training_variants;
- training_text;
- test_variants;
- test_text.

3.1 training_variants / test_variants

Este dataset está estruturado da seguinte maneira:

dfTrain_variants

Filter

	ID	Gene	Variation	Class
1	0	FAM58A	Truncating Mutations	1
2	1	CBL	W802*	2
3	2	CBL	Q249E	2
4	3	CBL	N454D	3
5	4	CBL	L399V	4
6	5	CBL	V391I	4
7	6	CBL	V430M	5
8	7	CBL	Deletion	1
9	8	CBL	Y371H	4
10	9	CBL	C384R	4
11	10	CBL	P395A	4
12	11	CBL	K382E	4
13	12	CBL	R420Q	4
14	13	CBL	C381A	4
15	14	CBL	P428L	5
16	15	CBL	D390Y	4
17	16	CBL	Truncating Mutations	1
18	17	CBL	Q367P	4
19	18	CBL	M374V	5
20	19	CBL	Y371S	4
21	20	CBL	H94Y	6

Figura 1 - Dataset training_variants / test_variants

- ID: corresponde ao ID único de cada observação;
- Gene: corresponde ao gene da mutação;
- Variation: aminoácido correspondente à mutação;
- Class: classe da mutação, variável objeto de estudo para previsão.

3.2 training_text / test_text

Este dataset está estruturado da seguinte maneira:

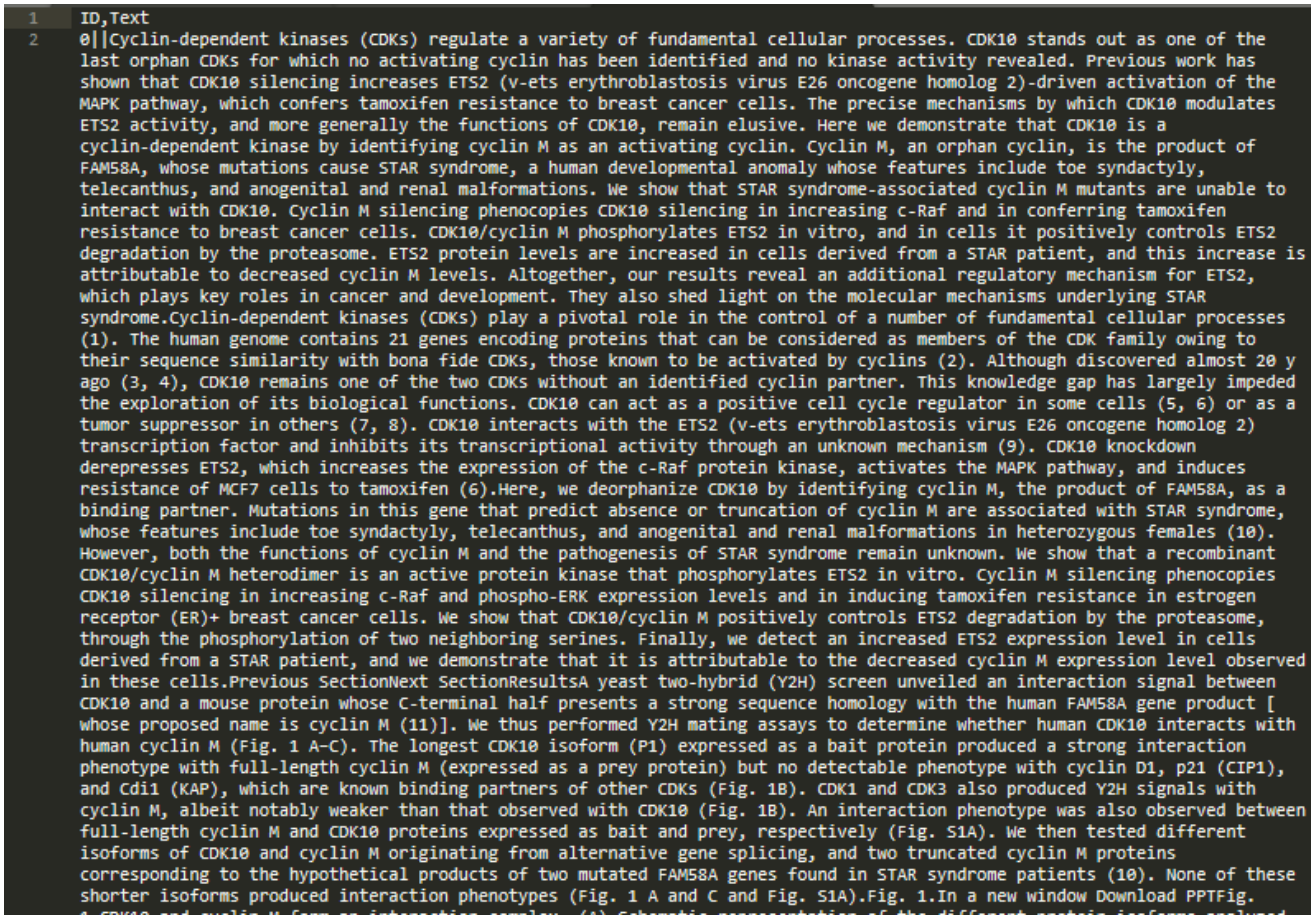

Figura 2 - Dataset training_text / test_text

- ID: corresponde ao ID único de cada análise patológica;
- Text: análise patológica usada para classificar cada mutação.

Este dataset possui como separador de colunas o caracter “||” (double pipe) pois como os textos das evidências clínicas possuem diversos tipos de caracteres, o double pipe conseguiu separar com sucesso cada evidência, porém a linguagem R possui uma limitação nas leituras convencionais de datasets, ela não aceita mais do que 1 caracter.

Deste modo, depois de muito pesquisar foi preciso realizar um parser (análise) em outro software substituindo o “||” por tabulação e assim foi possível ler esta base de conhecimento no RStudio.

Para consolidar, segue no próximo capítulo o Dicionário de Dados com a estruturação das variáveis.

4 – Dicionário de Dados

Variável	Significado
ID	ID da linha usada para ligar a mutação à evidência clínica
Gene	Gene onde a mutação genética está localizada
Variation	Aminoácidos alterados para essas mutações
Text	Evidência clínica usada para classificar a mutação genética
Class	1-9 classes em que a mutação genética foi classificada

Figura 3 - Dicionário de Dados

Temos ao todo 3.321 observações no dataset “training_variants”, ou seja, 3.321 linhas com as associações entre Gene, Variation e Class, porém no dataset “training_text” temos apenas 138 observações limitando nosso estudo em 138 linhas de evidências clínicas.

Para um Cientista de Dados isso pode ser decisivo no momento de criar modelos de Machine Learning pois quanto menos observações menos precisão na entrega do modelo final, porém não se deixe enganar, em cada uma das 138 linhas temos aproximadamente 40.000 informações. Basta observar a figura 2, que tem apenas 5% de uma linha à mostra. Isso comprova que não é um projeto trivial, e sim um projeto muito desafiador.

Nos capítulos seguintes daremos início à fase 1 do projeto que compete à análise exploratória utilizando técnicas de text mining para analisar textos e correlações entre palavras. Sem dúvida uma análise que vai trazer um entendimento de alto nível para o projeto.

A seguir vamos abordar as bibliotecas utilizadas.

5 – Bibliotecas utilizadas

Utilizar bibliotecas é imprescindível para o bom andamento de um projeto de Data Science. Neste projeto foram utilizadas as seguintes bibliotecas:

- readr[5]: biblioteca para leitura de datasets;
- tm[6]: framework para aplicações de text mining;
- textstem[7]: ferramentas para realizer Stemming e Lemmatization;
- dplyr[8]: facilitador de manipulação de dados;
- Rgraphviz[9]: interface para plotar objetos gráficos em R;
- ggplot2[10]: biblioteca utilizada para plotar gráficos;
- RColorBrewer[11]: biblioteca para alterar cor dos gráficos;
- qdapDictionaries[12]: dicionário de palavras para análises de textos com qdap;
- qdap[13]: automatiza processos de análises qualitativas em textos;
- scales[14]: biblioteca para auxiliar na transformação de escalas nos dados;
- e1071[15]: biblioteca com diversas funções para análise de dados, incluindo algoritmos de Machine Learning
- caret[16]: outra biblioteca de Machine Learning;
- ROCR[17]: pacote para auxiliar na construção de métricas de avaliação;
- nnet[18]: framework que contém o algoritmo de Redes Neurais Artificiais;
- rpart[19]: framework com algoritmo de Machine Learning de Árvores de Decisão;
- randomForest[20]: outro algoritmo de Machine Learning;
- xgboost[21]: algoritmo Extreme Gradient Boosting de Machine Learning.

[5] https://cran.r-project.org/web/packages/readr/index.html
[6] https://cran.r-project.org/web/packages/tm/index.html
[7] https://cran.r-project.org/web/packages/textstem/index.html
[8] https://www.rdocumentation.org/packages/dplyr/versions/0.7.8
[9] https://www.bioconductor.org/packages/release/bioc/html/Rgraphviz.html
[10] https://www.rdocumentation.org/packages/ggplot2/versions/3.2.1
[11] https://www.rdocumentation.org/packages/RColorBrewer/versions/1.1-2/topics/RColorBrewer
[12] https://cran.r-project.org/web/packages/qdapDictionaries/index.html
[13] https://cran.r-project.org/web/packages/qdap/index.html
[14] https://cran.r-project.org/web/packages/scales/index.html
[15] https://cran.r-project.org/web/packages/e1071/index.html
[16] http://topepo.github.io/caret/index.html
[17] https://cran.r-project.org/web/packages/ROCR/index.html
[18] https://cran.r-project.org/web/packages/nnet/index.html
[19] https://cran.r-project.org/web/packages/rpart/index.html
[20] https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest
[21] https://cran.r-project.org/web/packages/xgboost/index.html

6 – Feature Engineering

Feature Engineering, ou Engenharia de Atributos é o processo de tratamento, adição e remoção de variáveis.

Esse processo consiste em descobrir quais colunas de dados criam os atributos mais úteis para melhorar a precisão do modelo de aprendizagem de máquina. Identificar atributos bons e ruins é parte importante dando reflexo no resultado final, outra possibilidade é adicionar variáveis relevantes com base nos dados fornecidos.

Para trabalhar com textos é preciso transformar nosso dataset em um Corpus, ou seja, um conjunto de documentos com propriedades para análises linguísticas coletados criteriosamente para serem objeto de pesquisa.

Cada evidência clínica anotada por especialistas ao redor do mundo foi separada em um objeto corpus onde realizamos uma série de limpeza e transformações nas variáveis para que fosse possível entrar com a análise quantitativa, resumidamente foi desenvolvido:

- Remoção de caracteres especiais: - / @ | \\ _
- Transformação de todas as palavras para letras minúsculas: com a função “content_transformer(tolower)”;
- Remoção de números: com a função “removeNumbers”;
- Remoção de pontuações: com a função “removePunctuation”;
- Remoção de Stopwords (palavras que podem ser consideradas irrelevantes para o conjunto de resultados, como por exemplo: as, e, os, de, para);
- Ajuste de espaços em branco: com a função “stripWhitespace”;
- Lemmatization (transformação da palavra em seu lema, por exemplo: correndo, correu, correria, tudo sendo minimizado ao verbo correr);
- Stemming (remoção do prefixo e sufixo das palavras);
- Encoding (para evitar conflitos com softwares);
- Transformação do Corpus em Term Document Matrix (uma matriz que computa a quantidade que cada termo aparece dentro do texto, ou seja, a frequência e sua contribuição para o projeto)

A fase crítica do projeto foi o processo de transformação com a aplicação do Lemmatization e Stemming pois em primeiro momento utilizei apenas o Stemming removendo prefixos e sufixos das palavras acreditando que seria suficiente para obter um bom resultado mas no final do projeto pra minha surpresa foi a pior abordagem que podia ter escolhido retornando resultados insatisfatórios visto que palavras como “mutation”, “sequence”, “table” foram reduzidas à “mutat”, “sequenc”, “tabl” transformando completamente o entendimento do contexto.

7 – Análise Quantitativa

A análise exploratória é fundamental antes de realizar qualquer procedimento com os dados fornecidos pois é a partir dela que podemos entender onde temos melhores variáveis, onde temos problemas, onde temos oportunidades de melhoria, e dessa maneira podemos gerar uma série de conclusões importantes para dar continuidade no processo de aprendizado de máquina.

Durante a fase de análise exploratória vou inserir algumas comparações entre as abordagens com Lemmatization e Stemming provando que o Lemmatization é a maneira mais adequada de tratar as variações nas evidências clínicas deste problema de negócio.

O processo de feature engineering acima consumiu aproximadamente 60% do tempo da fase 1 do projeto pois ela impacta todo o restante das análises desenvolvidas.

7.1 Palavras mais frequentes encontradas nas evidências clínicas

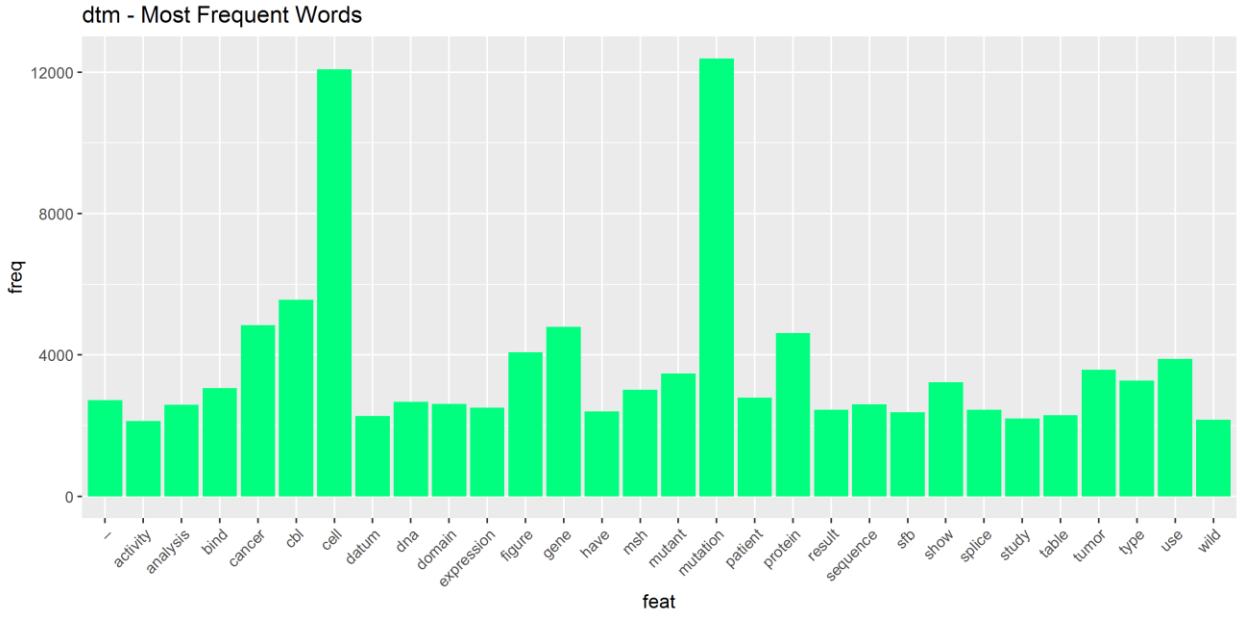

Figura 4 - Palavras mais frequentes com Lemmatization

Observe que “mutation”, “cell”, “cbl” (Casitas B-lineage Lymphoma), “cancer”, “gene”, “tumor” são as palavras que mais aparecem considerando todas as evidências clínicas anotadas pelos especialistas. Isto é um bom sinal pois todas as palavras estão relacionadas com o problema de negócio. Todo o tempo gasto no feature engineering deve ser considerado como um investimento e não como perda de tempo.

Apenas à nível de comparação vou apresentar o resultado trocando Lemmatization por Stemming. Observe na figura a seguir que palavras como “cbl” ou “tumor” nem aparecem entre as mais frequentes pois foram parcialmente eliminadas atrapalhando o entendimento do todo.

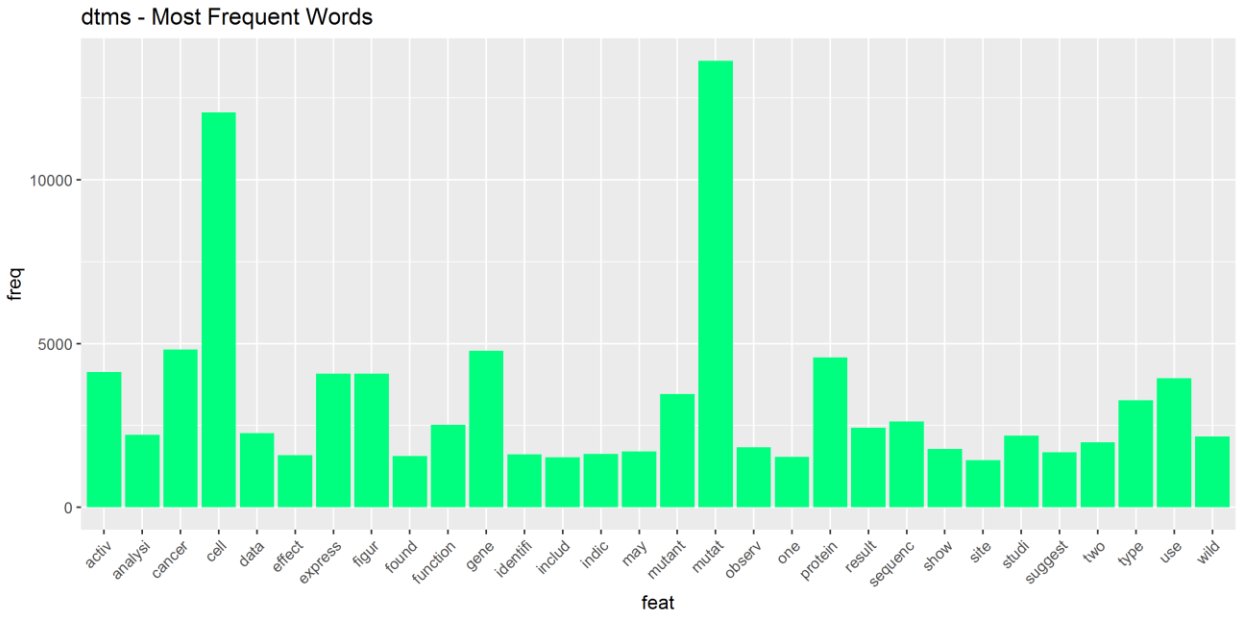

Figura 5 - Palavras mais frequentes com Stemming

Palavras como “mutation” foram reduzidas à “mutat”, “study” se tornou “studi”, dentre outras. Observe também que “activity” além de ter sido transformada para “activ” teve aumento na sua frequência de 2.000 para quase 5.000, isso pois provavelmente ela está sendo adicionada à outras palavras com significados semelhantes mas que na verdade ocupam posições de entendimento diferente no texto.

Para esse problema de negócio Lemmatization se mostrou mais coerente. Observe a seguir as palavras menos frequentes.

7.2 Palavras menos frequentes encontradas nas evidências clínicas

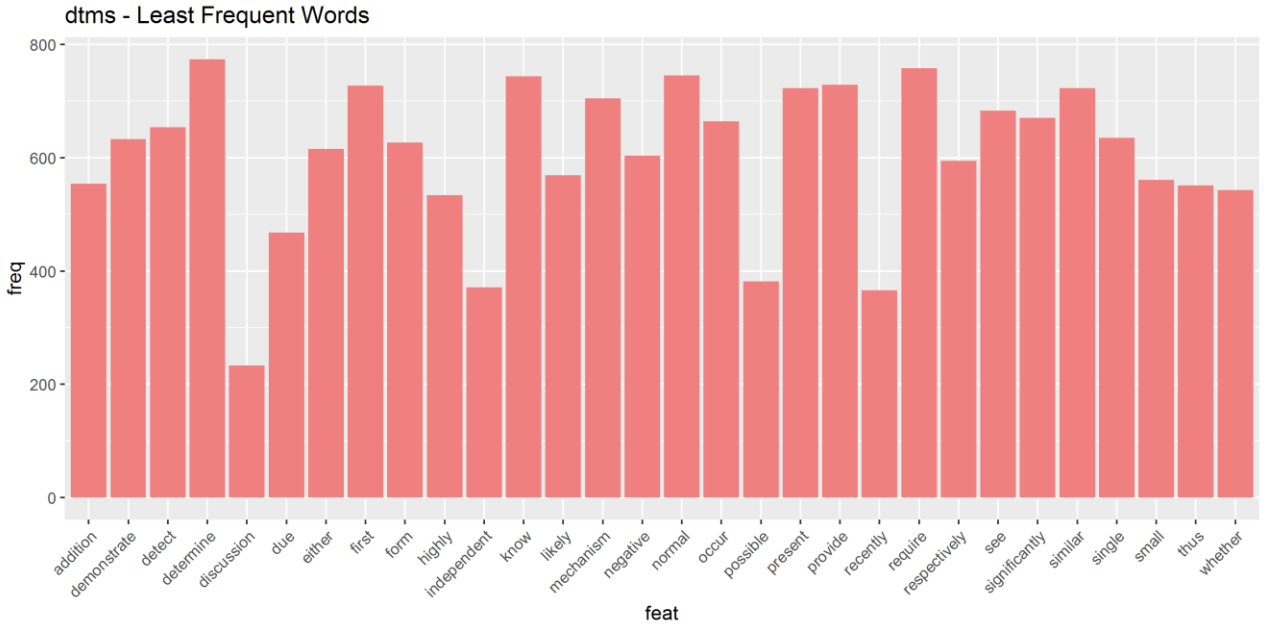

Figura 6 - Palavras menos frequentes com Lemmatization

Palavras menos frequentes como “discussion”, “possible”, “recently” foram encontradas porém isso não significa que são menos importantes para prever uma ocorrência de mutação genética que ocasiona câncer em pacientes. É preciso ter cuidado com esta análise pois aqui a análise é apenas qualitativa e não preditiva.

Dentro da nossa base de conhecimento para encontrar as palavras mais relevantes que fazem com que encontremos a classe correta do câncer detectado será automatizada através dos algoritmos de Machine Learning, ou seja, além de utilizar os algoritmos para prever a classe correspondente, vou usá – los para me informar quais palavras seriam as mais relevantes no todo entrando inclusive na especificação de cada classe mostrando quais palavras ajudam a prever cada tipo de câncer. Uma abordagem diferente e poderosa que certamente traz grande valor ao projeto.

7.3 Wordcloud de palavras com frequência maior de 300

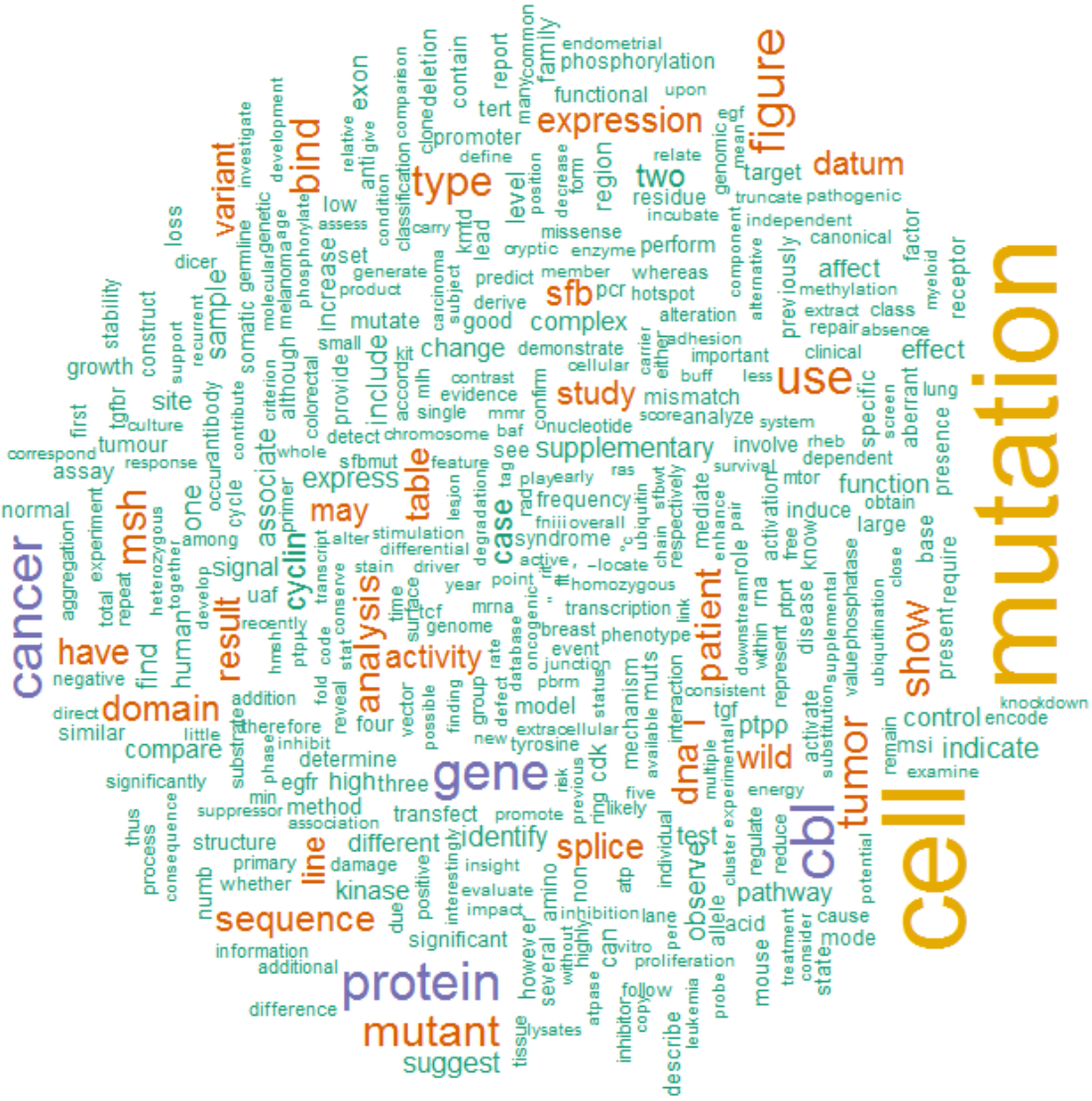

Figura 7 - Wordcloud

Uma wordcloud (ou nuvem de palavras) é um conglomerado de palavras que aparecem em diferentes tamanhos indicando que a maior entre elas aparece com mais frequência e a menor entre elas com menos frequência.

Observe que palavras como “mutation”, “cell”, “cancer”, “gene”, “protein”, “dna”, “tumor”, “sfb”, “msh” são bons indicadores de que estamos no caminho certo.

Durante a análise quantitativa o que queremos, além de olhar para as palavras mais frequentes, é encontrar o relacionamento entre palavras, ou seja, se duas ou mais palavras aparecem juntas no texto e com que frequência aparecem juntas gerando assim insights valiosos. Por esse motivo provavelmente aqui seria o momento de uma reunião com os especialistas patológicos a fim de entender o que consideram como sendo mais relevante para determinar uma mutação genética que ocasione o câncer.

Como aqui estou realizando a análise através de uma base de conhecimento a qual não tenho acesso aos especialistas, vou usar palavras chave e assim analisar as correlações com outros termos identificando quais geram mais proximidade com a nossa problemática.

7.4 Procurando por associações entre palavras

Vamos iniciar com a palavra foco do estudo:

1. Cancer

```
> findAssocs(dtm, "cancer", corlimit = 0.66)
$cancer
         breast           study          sweden             may         allelic 
           0.71            0.70            0.70            0.69            0.69 
      frequency      occurrence        supplier epidemiological         triplet 
           0.69            0.69            0.69            0.69            0.69 
    somatically         tubular            risk       ethnicity    polymorphism 
           0.68            0.68            0.67            0.67            0.67 
  susceptibility   environmental       penetrant       recessive        nontumor 
           0.67            0.67            0.67            0.67            0.67 
   northwestern           pires             tgc   heterogeneity            meta 
           0.67            0.67            0.67            0.66            0.66
```

Figura 8 - Associações com "cancer"

As palavras que mais aparecem juntas com câncer são:

- breast: um tipo comum de câncer entre mulheres, o câncer de mama (breast cancer)
- study: estudo de fenômenos cancerígenos
- allelic: Os alelos são as formas alternativas, gerados por mutações, de um mesmo gene

Através da análise de correlação com a palavra “câncer” podemos ver que provavelmente a nossa base de conhecimento esteja relacionada ao estudo do câncer de mama e a frequência de ocorrência da heterogeneidade alélica, ou seja, presença de diferentes alelos mutantes em um lócus.

Como temos indicações de que estamos lidando com um tipo de câncer de mama (breast cancer) vamos entender quais palavras estão correlacionadas com este tipo.

2. Breast

```
> findAssocs(dtm, "breast", corlimit = 0.64)
$breast
 susceptibility          sweden    geographical   questionnaire        familial
           0.85            0.82            0.81            0.81            0.74
epidemiological          cancer        mediator      occurrence       polygenic
           0.72            0.71            0.71            0.70            0.70
           risk     minneapolis       recessive            cdca     eligibility
           0.68            0.68            0.67            0.67            0.67
            sas         tubular    heterozygote           pires            live
           0.67            0.67            0.66            0.66            0.65
           york        supplier       hydroxide            meta          pasche
           0.65            0.65            0.65            0.65            0.65
           brca         finland             tgc
           0.64            0.64            0.64
```

Figura 9 - Associações com "breast"

A primeira palavra com 85% de correlação com breast é “suscetibilidade”, ou seja, alta disposição de contrair enfermidades. Temos também “cancer” com 71% de correlação indicando também que estas palavras costumam aparecer juntas nas evidências clínicas.

Termos técnicos como “recessive”, “cdca”, “heterozygote”, “hydroxide”, “tgc” corroboram o estudo do câncer.

Mas o que chamou atenção nas associações com “breast” é que temos evidências geográficas em questão como “Sweden”, “Minneapolis”, “York”, “Finland”. Realizando uma pesquisa rápida é possível identificar que são centros de estudo e tratamento de câncer, possíveis locais onde se encontram especialistas que ajudaram a montar nossa base de conhecimento.

Para algumas referências deixo em anexo os sites:

- The Swedish Cancer Society[22]
- American Cancer Society[23]
- York Against Cancer[24]
- Cancer Society of Finland[25]

[22] https://www.cancerfonden.se/om-oss/about
[23] https://www.cancer.org/treatment/support-programs-and-services/patient-lodging/hope-lodge/minneapolis.html
[24] https://www.yorkagainstcancer.org.uk/home
[25] https://www.cancersociety.fi/

3. Cbl (Casitas B-lineage Lymphoma)

```
> findAssocs(dtm, "cbl", corlimit = 0.72)
$cbl
         ubiquitin             ligase             sodium               cruz
              0.89               0.88               0.83               0.81
             santa             disomy             finger     ubiquitination
              0.80               0.80               0.80               0.80
       uniparental            adaptor             linker      biotechnology
              0.80               0.79               0.79               0.78
myeloproliferative          conjugate          overnight            resolve
              0.77               0.76               0.76               0.76
              cold                rtk              twice         supplement
              0.75               0.75               0.75               0.74
              wash        endocytosis      densitometric            cytokine
              0.74               0.74               0.73               0.72
             tween             solely
              0.72               0.72
```

Figura 10 - Associações com "cbl"

As correlações acima são na sua grande maioria termos técnicos relacionados à área da saúde que estudam componentes ligados à um tipo de câncer, o Cbl. De acordo com My Cancer Genome[26] (organização que disponibiliza informações sobre diferentes tipos de câncer) “Cbl (nomeado após Casitas B-lineage Lymphoma) é um gene de mamífero que codifica a proteína CBL, que é uma ubiquitina-proteína ligase E3 envolvida na sinalização celular e ubiquitinação de proteínas. Mutações nesse gene foram implicadas em vários cânceres humanos, particularmente leucemia mieloide aguda.”

A cada passo adiante vemos que o processo de limpeza e transformação nos dados foi crucial sendo um investimento para gerar insights valiosos no decorrer do projeto. A associação acima prova isso, pois foram encontradas evidências em documentos oficiais (My Cancer Genome) de que a maior parte das palavras correlacionadas com “cbl” estão envolvidas com um tipo de mutação que gera câncer em humanos. Basta observar as palavras sublinhadas e as suas correlações.

7.5 Palavras que aparecem juntas e suas interligações

Até o momento conseguimos entender visualmente quais as palavras mais frequentes, ou seja, uma análise univariada pois olhamos apenas para as aparições e não as correlações, em seguida procuramos por associações entre palavras, ou seja, adicionamos um pouco mais de inteligência ao projeto pois sabemos quais palavras estão mais próximas umas das outras identificando que as evidências clínicas estão indicando um estudo do câncer de mama.

[26] https://www.mycancergenome.org/

Vamos dar mais um passo, um grande passo, adicionando agora mais uma camada de inteligência indicando todas as correlações entre palavras e suas interligações com outras palavras. A visualização é fantástica pois justando os parâmetros corretos podemos obter insights valiosos adicionando ainda mais valor ao projeto.

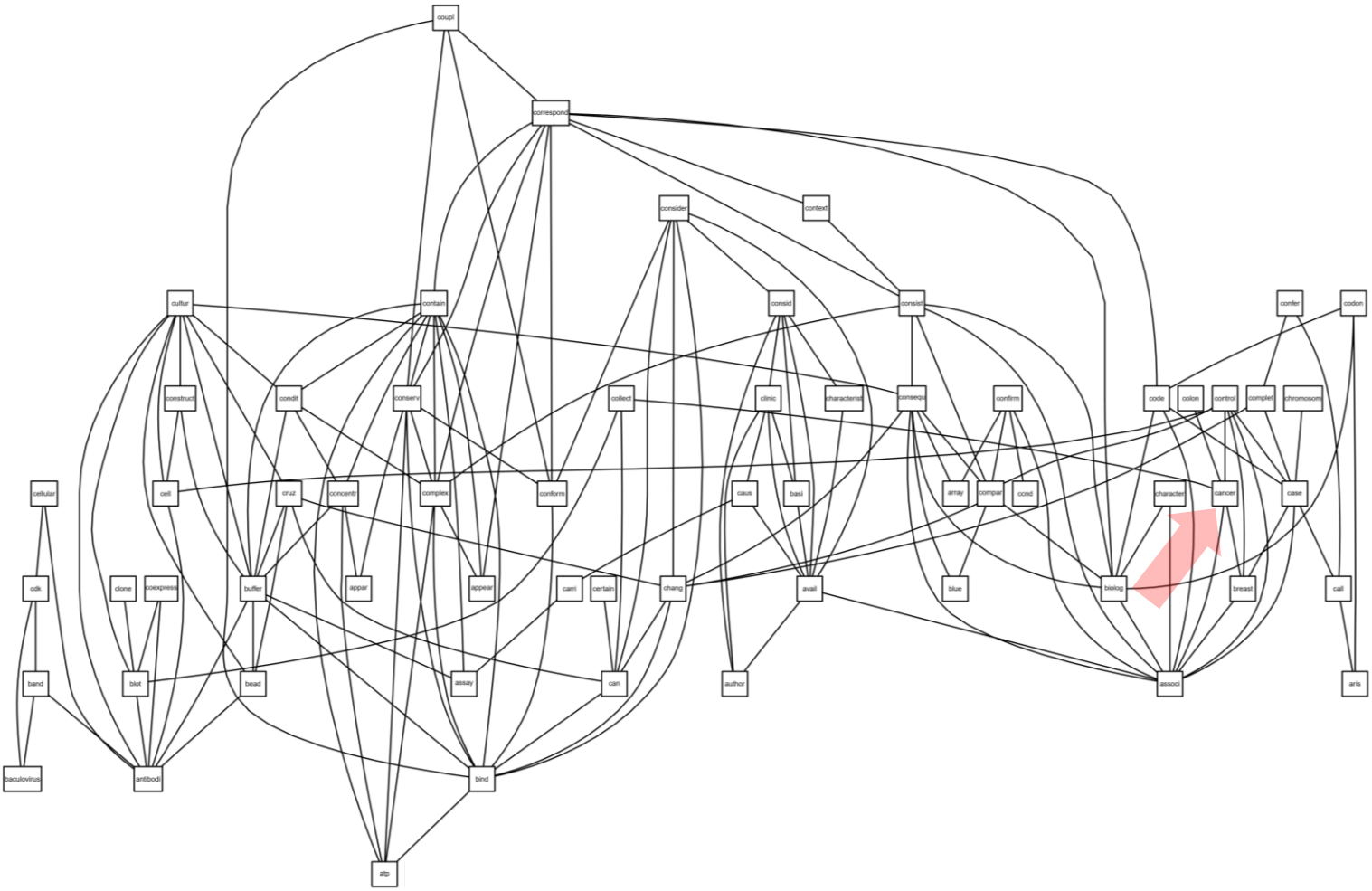

Figura 11 - Interligações entre as palavras

Como temos muitas informações, aconselho que aproxime e verifique as interligações na figura acima. Observe no canto direito que a palavra “câncer” está interligada com “breast” que consequentemente está interligada com “case”, “control” e “association”, ou seja, estas palavras estão aparecendo com mais frequência juntas gerando inteligência à análise. Essa interligação indica o estudo e controle de casos de câncer de mama.

Poderíamos realizar uma série de estudos dentro das interligações, com diferentes correlações, diferentes espaços amostrais, diferentes palavras, enfim, infinitas possibilidades. Por isso é muito importante que os especialistas patológicos caminhem em proximidade com este projeto permitindo o entendimento das melhores interligações entre palavras a fim de classificar cada tipo de câncer.

Nossa última análise, da parte I deste projeto, será identificar como está a distribuição de letras dentre as palavras em nossa base de conhecimento. Uma abordagem diferente e interessante.

7.6 Distribuição de Letras

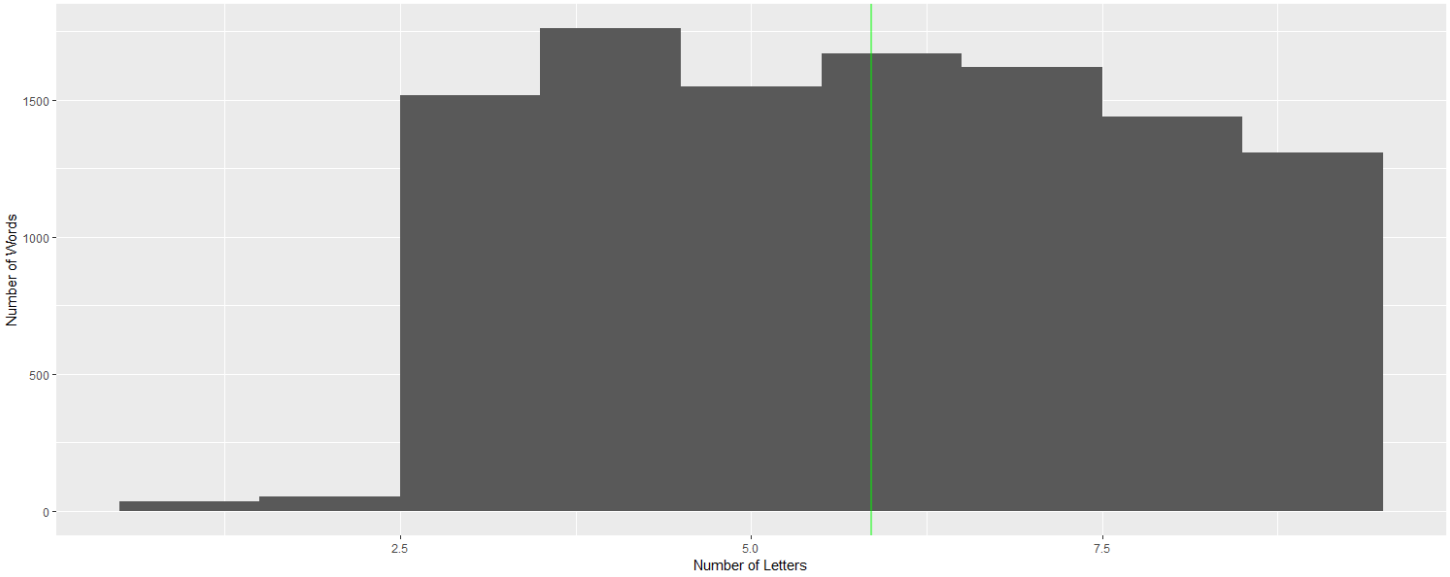

Figura 12 - Letras x Palavras

A figura acima apresenta a distribuição de letras versus a quantidade de palavras com aquela quantidade de letras. Temos muito mais palavras que possuem de 3 à 7 letras em nossa base de conhecimento, quase nenhuma com 1 ou 2 letras e média (linha verde) de 6 letras por palavras.

Para nossos modelos preditivos essa informação pode ser determinante visto que aumentamos a complexidade ao aumentar a quantidade de letras. Ao analisar a esparsidade da base de conhecimento convertida em matriz termo – documento estamos com 93% de termos esparsos, uma alta quantidade onde dificilmente os modelos de Machine Learning vão convergir com resultados satisfatórios, pensando nisso decidi reduzir a esparsidade à 20% e dar continuidade no projeto reduzindo também, de maneira considerável, o custo de processamento.

Sendo assim, a partir do próximo capítulo damos início à fase II do projeto, onde vamos construir 5 modelos de Machine Learning e analisar individualmente suas acurácias e como cada palavra ajuda a prever cada classe.

8 – Machine Learning

Antes de criarmos os modelos preditivos, vamos estudar a quantidade de dados que temos em cada classe que queremos realizar previsões:

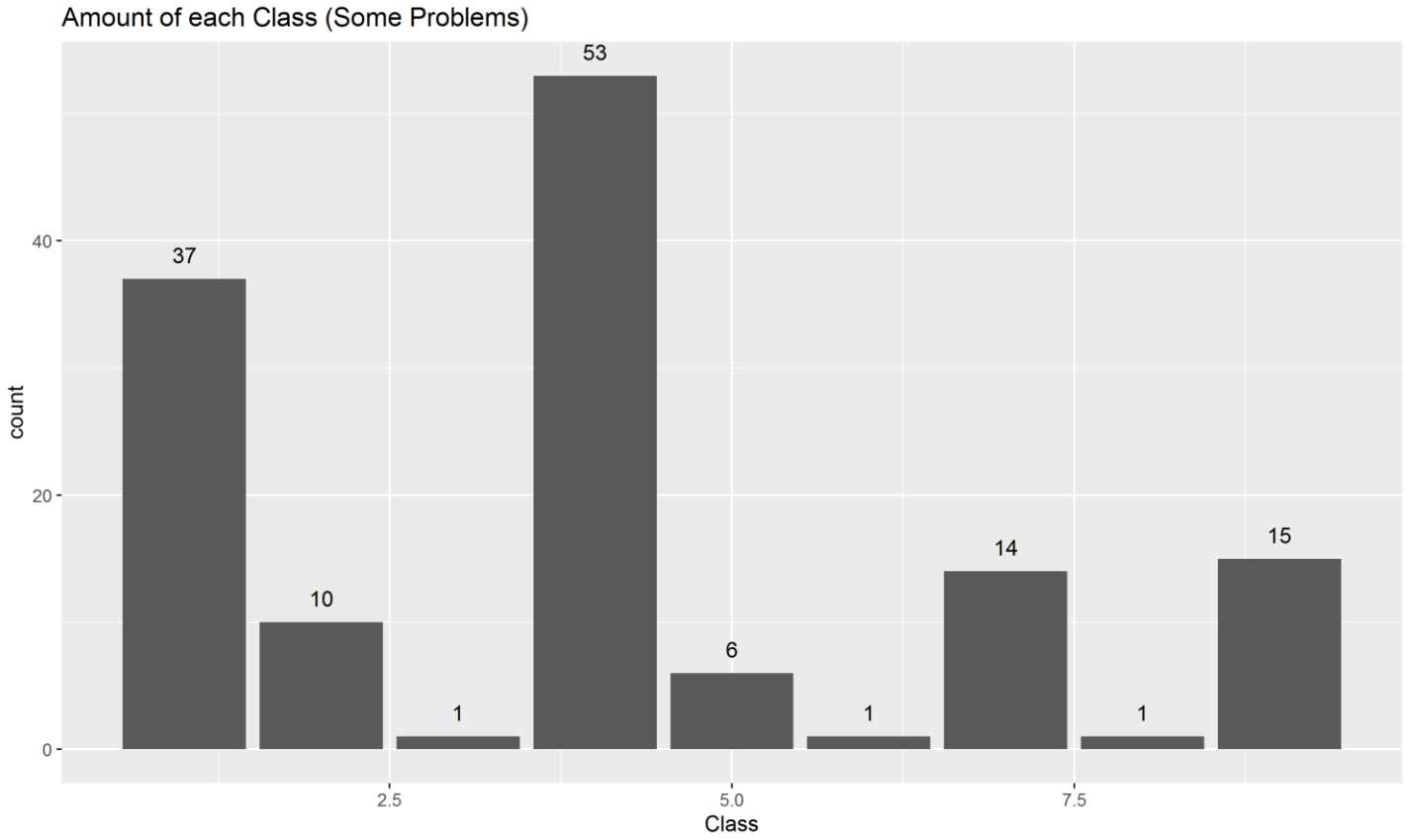

Figura 13 - Distribuição de Classes

8.1 Troubleshotting

Aqui claramente temos um problema, 3 das 9 classes possuem apenas uma observação. Como posteriormente vou dividir minha base de conhecimento em 60% dos dados para treino e 40% dos dados para teste, fatalmente essas classes (3, 6 e 8) ficarão em treino OU em teste. Se ficar em treino, quando eu for avaliar meus modelos através de uma Confusion Matrix receberei mensagens de erro pois não haverão dados para comparações; e se ficar apenas em teste, vou treinar os modelos sem estas classes e no momento de avaliação terei aumento no erro visto que o modelo não sabe que deve prever aquela classe.

O ideal seria realizar adições de variáveis dummy (ou buscar mais dados na fonte) referente às classes minoritárias, porém como este não é um problema de negócio trivial e cada classe possui por linha um conjunto de dados com mais de 40.000 palavras anotadas por especialistas, foi preferível remover estar classes.

8.2 Distribuição de Classes

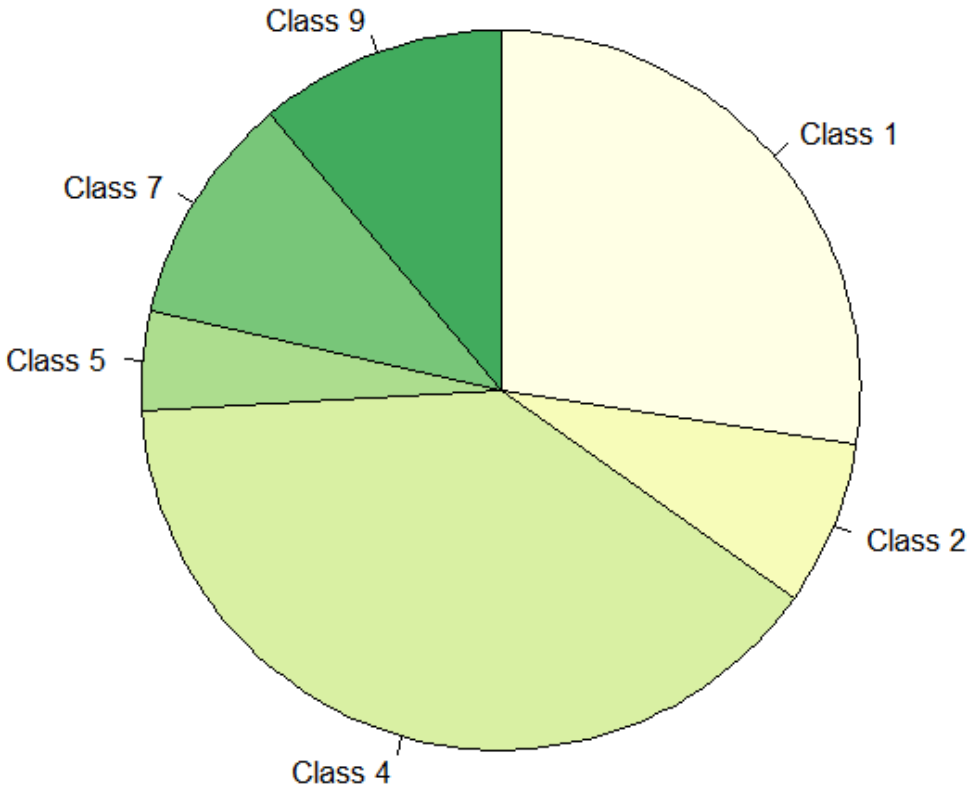

Figura 14 - Distribuição de Classes

Agora sim temos o conjunto de dados ajustados para as próximas etapas de Machine Learning tendo as classes 1 e 4 como dominantes em nossa base de conhecimento, representando 27% e 39% do todo, respectivamente.

8.3 Neural Network

O primeiro modelo que iremos explorar será o de Redes Neurais Artificiais e como o nome sugere é um modelo análogo ao sistema neural humano sendo reproduzido em máquinas.

Assim como neurônios no sistema nervoso de humanos que se interconectam, uma rede neural artificial é construída através de interconexões artificiais onde cada neurônio realiza sua ativação através de pesos e funções pré-determinadas. O forte desta abordagem está no processamento paralelo de informações permitindo lidar com dados não – lineares sendo indicado para encontrar padrões dentro de datasets complexos, como o deste projeto.

Apesar de muito útil em casos complexos, uma rede neural artificial é considerada um caixa preta pois as únicas camadas que conhecemos é a de entrada e saída, o que acontece entre elas é um grande emaranhado de interconexões desconhecidas mas que convergem para o resultado final. A figura a seguir representa uma rede neural artificial.

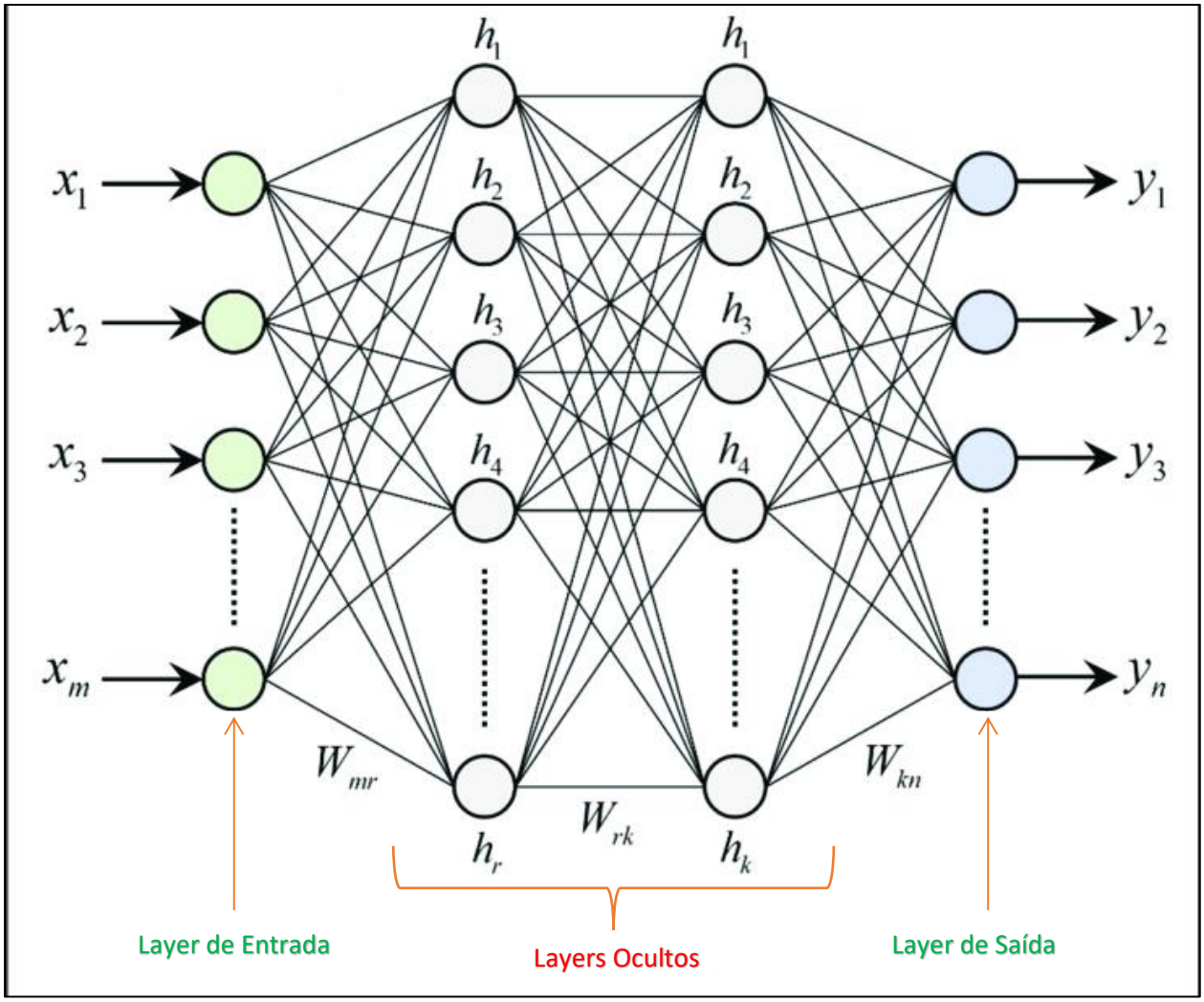

Figura 15 - Rede Neural Artificial

Conforme figura acima, as interconexões dos layers ocultos são desconhecidas e as informações são transmitidas entre neurônios de forma análoga aos neurônios humanos onde funções de ativações biológicas permite a passagem ou não de determinado estímulo.

Em Redes Neurais Artificiais, ao receber uma informação de entrada, conhecendo a saída, os layers ocultos se ajustam realizando backpropagation ajustando seus pesos automaticamente a fim de alcançar a máxima eficiência no resultado de saída.

Em nosso modelo utilizamos todas as variáveis como entrada e informamos as classes de saída desejada obtendo assim as seguintes métricas:

Erro Residual -> 3.5090
AIC -> 643.509

8.3.1. Confusion Matrix e Acurácia

Confusion Matrix é um dos muitos medidores de performance para avaliar os dados previstos por um modelo de machine learning. Em problemas de classificação, com a confusion matrix podemos ver claramente os dados que foram classificados corretamente e incorretamente obtendo assim a acurácia do modelo. Resumidamente uma Confusion Matrix pode nos informar o que aconteceu com todos os dados classificados. Todos os modelos que vamos criar serão avaliados a partir de uma confusion matrix entre valores esperados versus valores previstos.

Segue confusion matrix após apresentados dados de teste ao modelo de Redes Neurais Artificiais:

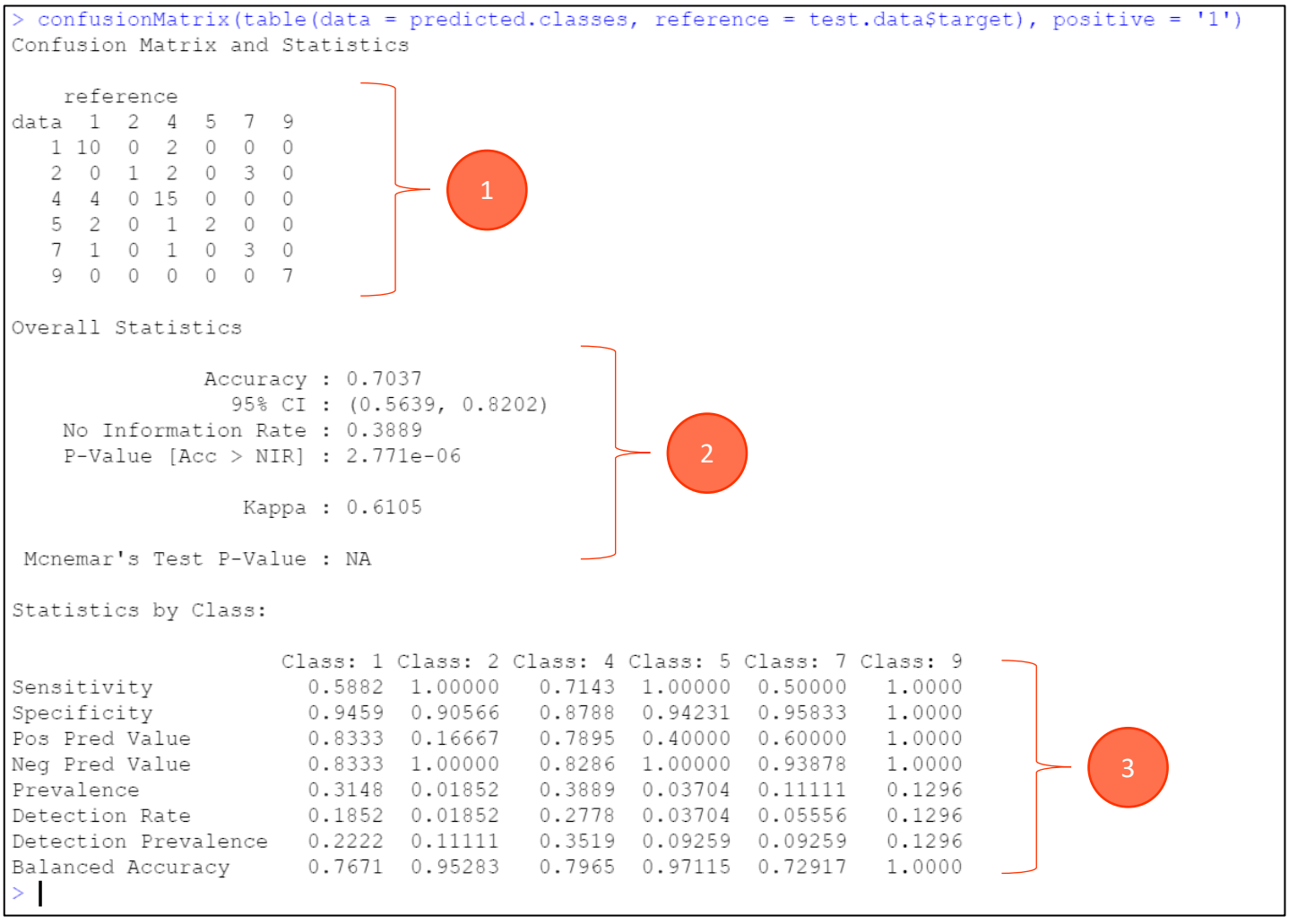

Figura 16 - Confusion Matrix Neural Network

Vamos dividir as informações acima em três etapas a fim de consolidar o entendimento do resultado.

→ Métricas Nº 1

Neste item da figura 16 temos a confusion matrix em questão onde:

- reference: corresponde aos dados esperados e apresentados como teste ao modelo
- data: corresponde aos dados previstos pelo modelo

Vamos usar como exemplo a classe 9, observe que de 7 valores previstos o modelo acertou todas as previsões. Um resultado ótimo, diferente da classe 7 que de 6 valores apresentados acertou 3 e classificou erroneamente os outros 3 considerando como se fossem da classe 2.

As nossas classes majoritárias (classes 1 e 4) apresentaram boa performance também acertando a maior parte dos resultados previstos.

→ Métricas Nº 2

Neste item da figura 16 temos a acurácia do modelo quando apresentados dados desconhecidos:

Acurácia (Neural Network) -> 70.37%

Conforme vimos na métrica nº 1 a maior parte das classes foram corretamente classificadas, e a acurácia vem corroborar com mais um indicativo de boa performance chegando ao máximo de 70.37% de precisão. Durante a apresentação dos resultados de cada modelo preditivo vamos comparar as métricas e analisar qual oferece melhor precisão.

→ Métricas Nº 3

Neste item da figura 16 temos métricas estatísticas indicando como cada classe se comporta em determinada análise. Podemos destacar:

- Balanced Accuracy: classes 2, 5 e 9 com ótimos resultados próximo de 100%
- Specificity: todas as classes tiveram ótimas métricas neste quesito que determina quão apto o modelo está para previsões corretas. Isso mostra como redes neurais são poderosos estimadores.

Estudamos no início da análise qualitativa quais eram as palavras mais frequentes e quais estavam mais correlacionadas, porém isso não significa que são as palavras mais relevantes para que o modelo preditivo encontre as classes corretas.

8.3.2. Palavras mais importantes de acordo com o modelo preditivo

Através de um objeto de Cross – Validation utilizaremos todas as combinações possíveis entre variáveis realizando um processo iterativo com o modelo de Machine Learning resultando nas variáveis mais importantes para esse modelo.

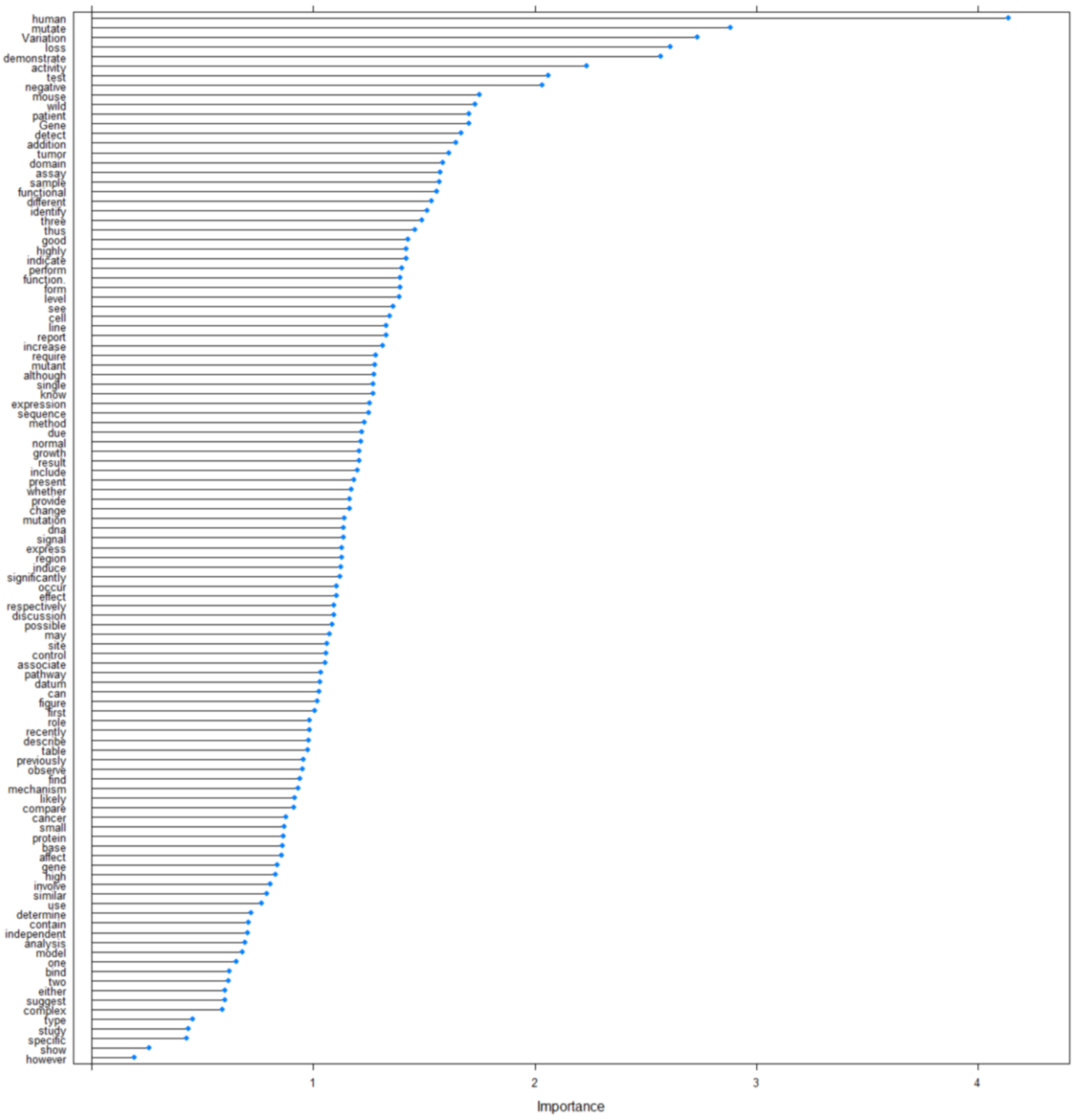

Figura 17 - Palavras Mais Importantes (Neural Network)

De acordo com o modelo, as palavras mais relevantes presentes nas evidências clínicas para prever as classes apresentadas são:

- human, mutate, Variation, loss, demonstrate, activity, tumor

8.4 Decision Tree

Nosso segundo modelo será Decision Tree, ou Árvore de Decisão, que analogamente à uma Árvore utiliza raízes e galhos para pecorrer o conjunto de dados e escolher a melhor combinação de elementos fornecendo a melhor precisão possível.

Imagine uma árvore invertida:

Figura 18 - Decision Tree

Nós de término (ou folhas) estão na parte inferior da árvore de decisão. Isto significa que as árvores de decisão são desenhadas de cabeça para baixo. Dessa forma, as folhas são o fundo e as raízes são os topos (figura acima).

Uma Árvore de Decisão trabalha tanto com variáveis categóricas como contínuas, e funciona com a divisão da população (ou amostra) em subpopulações (dois ou mais conjuntos) baseando-se nos divisores mais significativos das variáveis de entrada. Por esse e outros tantos motivos árvores de decisão são utilizadas em problemas de classificação e regressão onde o algoritmo de aprendizagem supervisionada possui uma variável alvo pré-definida.

Estes modelos apesar de simples possuem alto poder preditivo o que pode gerar um problema de overfitting, onde aprendem demais e oferecem pouca precisão em novos e desconhecidos dados. É preciso saber ajustar e possivelmente podar as Árvores de Decisão.

Com a utilização de todas as variáveis modelamos uma Árvore de Decisão, resultando em:

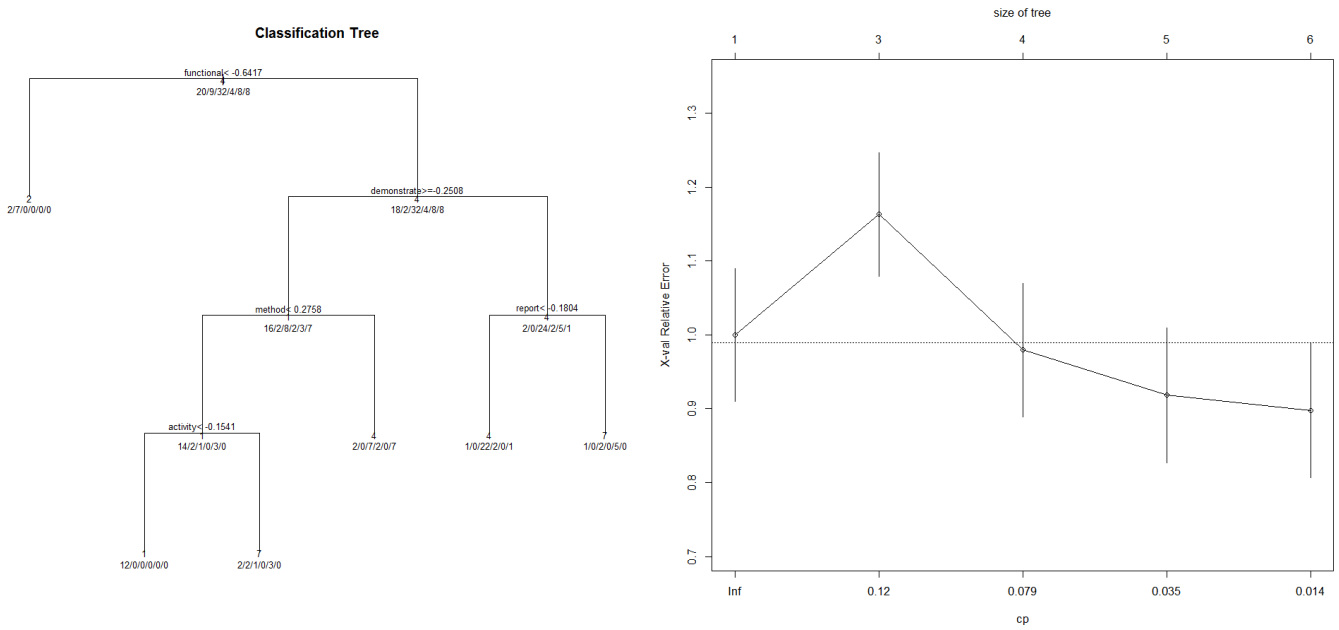

Figura 19 - Decision Tree e Erro Relativo

Na figura acima temos à esquerda nossa Árvore de Decisão com 4 camadas e na direita temos a apresentação do tamanho de cada árvore construída e o erro relativo. Conforme aumentamos a complexidade da nossa árvore de decisão podemos ver que o erro cai gradativamente até um ponto de estabilidade, ou seja, apesar de uma árvore de tamanho 3 ter apresentado uma tendência de aumento no erro relativo, ao treinar uma árvore com 4 camadas temos uma relação ótima entre erro e complexidade evitando overfitting e não precisando podar esta árvore.

8.4.1. Confusion Matrix e Acurácia

```
> confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')
Confusion Matrix and Statistics

    reference
data  1  2  4  5  7  9
   1  7  0  3  0  0  1
   2  0  1  3  0  1  1
   4  3  0 12  2  0  5
   5  0  0  0  0  0  0
   7  7  0  3  0  5  0
   9  0  0  0  0  0  0

Overall Statistics

               Accuracy : 0.463
                 95% CI : (0.3262, 0.6039)
    No Information Rate : 0.3889
    P-Value [Acc > NIR] : 0.1641

                  Kappa : 0.2787

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 4 Class: 5 Class: 7 Class: 9
Sensitivity            0.4118  1.00000   0.5714  0.00000  0.83333   0.0000
Specificity            0.8919  0.90566   0.6970  1.00000  0.79167   1.0000
Pos Pred Value         0.6364  0.16667   0.5455      NaN  0.33333      NaN
Neg Pred Value         0.7674  1.00000   0.7188  0.96296  0.97436   0.8704
Prevalence             0.3148  0.01852   0.3889  0.03704  0.11111   0.1296
Detection Rate         0.1296  0.01852   0.2222  0.00000  0.09259   0.0000
Detection Prevalence   0.2037  0.11111   0.4074  0.00000  0.27778   0.0000
Balanced Accuracy      0.6518  0.95283   0.6342  0.50000  0.81250   0.5000
>
```

Figura 20 - Confusion Matrix Decision Tree

→ Métricas Nº 1

Diferente do modelo anterior (Neural Networks) a Árvore de Decisão apresentou resultados muito precários, observe que a classe 9 nem conseguiu ser prevista sendo confundida com as classes 1, 2 e 4.

A classe 5, por exemplo, previu todos os valores apresentados como sendo classe 4, porém observe que para as classes majoritárias (classe 1 e 4), onde temos mais dados para treinar o modelo, conseguimos melhores resultados prevendo as classes corretas, mas ainda assim não é um modelo ótimo.

→ Métricas Nº 2

A acurácia do modelo quando apresentados dados desconhecidos:

Acurácia (Decision Tree) -> 46.30%

Conforme vimos na métrica nº 1 a maior parte das classes foram erroneamente classificadas, e a acurácia vem corroborar com mais um indicativo de performance precária chegando ao máximo de 46.30% de precisão.

→ Métricas Nº 3

Vejamos como cada classe se comporta estatisticamente:

- Balanced Accuracy: apenas a classe 2 teve o melhor resultado nesta métrica, o que é demonstrado na confusion matrix pela previsão correta em 100%
- Sensitivity: quesito que determina quão apto o modelo está para previsões corretas demonstra que nem a classe majoritária 1 tem bom desempenho. Isso mostra como redes neurais são poderosos estimadores.

Apesar de ter apresentado um resultado ruim, é possível melhorar a performance deste modelo através da união de várias Árvores de Decisão formando uma Floresta Aleatória, ou RandomForest.

O randomForest faz parte da família de algoritmos que utilizam Métodos Emsemble para construção de modelos robustos. Vamos estudar o randomForest mais adiante.

8.4.2. Palavras mais importantes de acordo com o modelo preditivo

De acordo com o modelo, as palavras mais relevantes presentes nas evidências clínicas para prever as classes apresentadas são:

- analysis, demonstrate, include, functional, method, cell

8.5 K – Nearest Neighbors

Após um segundo modelo de Machine Learning precário, vamos explorar o algoritmo K – Nearest Neighbors (KNN).

O KNN é um algoritmo de classificação não – paramétrico e que utiliza o conceito de algoritmo lazy (preguiçoso) sendo um modelo de Machine Learning simples. Dizemos que é um algoritmo não – paramétrico pois ele não realiza fortes suposições em funções de mapeamento e desta maneira o KNN se torna livre para aprender a função (fórmula) que leva à previsões corretas a partir dos dados de treino. Métodos não – paramétricos são ótimos para bases de conhecimento com muitos dados e pouco conhecimento prévio.

Algoritmos do tipo lazy (preguiçosos) implicam em não generalizar uma função de mapeamento a partir dos dados de treino, na verdade o que eles fazem é capturar todos dados de treino e guardar para o momento de teste sem necessariamente realizar a fase de treinamento pois quando uma nova observação é apresentada ao modelo ele adere estes dados à classe dos k vizinhos mais próximos. Isso faz com que o algoritmo seja rápido na fase de treino porém exige um pouco mais de tempo em cada previsão.

```
> # Looking at each k value and the optimun k
> model_v3
k-Nearest Neighbors

 81 samples
109 predictors
  6 classes: '1', '2', '4', '5', '7', '9'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 72, 74, 72, 72, 74, 73, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.5503175  0.34846243
   7  0.5791138  0.39199033
   9  0.5244841  0.30417686
  11  0.4651323  0.21201284
  13  0.4663889  0.20566041
  15  0.4323280  0.14624309
  17  0.4094048  0.09433433
  19  0.3838095  0.05017136
  21  0.4104630  0.07943976
  23  0.4129101  0.06320521
  25  0.4207804  0.07290353
  27  0.4076852  0.04067481
  29  0.4118519  0.03992637
  31  0.4155556  0.04275258
  33  0.4072222  0.02515866
  35  0.4188889  0.04296205
  37  0.4020899  0.01176471
  39  0.4110185  0.02631640
  41  0.3979233  0.00000000
  43  0.3979233  0.00000000

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 7.
> # The best k:
> model_v3[["finalModel"]][["k"]]
[1] 7
> # with maximum accuracy of:
> max(model_v3[["results"]][["Accuracy"]])
[1] 0.5791138
```

Figura 21 - K - Nearest Neighbors Model

Conforme figura 21 conseguimos identificar que o melhor valor de k = 7 para acurácia de 0.5791. Melhorou quando comparado ao modelo de Árvore de Decisão, porém ainda não superou o modelo de Redes Neurais.

8.5.1. Confusion Matrix e Acurácia

Segue confusion matrix após apresentados dados de teste ao modelo:

```
> confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')
Confusion Matrix and Statistics

    reference
data  1  2  4  5  7  9
   1  9  0  3  0  0  0
   2  0  0  0  0  1  0
   4  6  1 18  2  5  2
   5  0  0  0  0  0  0
   7  0  0  0  0  0  1
   9  2  0  0  0  0  4

Overall Statistics

               Accuracy : 0.5741
                 95% CI : (0.4321, 0.7077)
    No Information Rate : 0.3889
    P-Value [Acc > NIR] : 0.004455

                  Kappa : 0.3628

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 4 Class: 5 Class: 7 Class: 9
Sensitivity            0.5294  0.00000   0.8571  0.00000  0.00000  0.57143
Specificity            0.9189  0.98113   0.5152  1.00000  0.97917  0.95745
Pos Pred Value         0.7500  0.00000   0.5294      NaN  0.00000  0.66667
Neg Pred Value         0.8095  0.98113   0.8500  0.96296  0.88679  0.93750
Prevalence             0.3148  0.01852   0.3889  0.03704  0.11111  0.12963
Detection Rate         0.1667  0.00000   0.3333  0.00000  0.00000  0.07407
Detection Prevalence   0.2222  0.01852   0.6296  0.00000  0.01852  0.11111
Balanced Accuracy      0.7242  0.49057   0.6861  0.50000  0.48958  0.76444
>
```

1

2

3

Figura 22 - Confusion Matrix (K - Nearest Neighbors)

→ Métricas Nº 1

Apenas as classes majoritárias tiveram um desempenho coerente, todas as outras infelizmente não conseguiram convergir com sucesso aos valores previstos.

→ Métricas Nº 2

A acurácia do modelo quando apresentados dados desconhecidos:

Acurácia (KNN) -> 57.41%

Conforme vimos na métrica nº 1 a maior parte das classes foram erroneamente classificadas

Aqui a acurácia melhorou quando comparada ao modelo anterior, mas não o suficiente para poder se tornar o modelo escolhido neste projeto. Mais um indicativo de performance precária chegando ao máximo de 57.41% de precisão.

→ Métricas Nº 3

Vejamos como cada classe se comporta estatisticamente:

- Balanced Accuracy: apenas 2 das 6 classes conseguiram ultrapassar 50% nesta métrica
- Sensitivity: quesito que determina quão apto o modelo está para previsões corretas demonstra que apenas a classe majoritária 4 conseguiu um resultado aceitável. Vale ressaltar que outras 3 classes chegaram à 0%.

Dificilmente este seria o modelo mais indicado para o nosso conjunto de dados, primeiro por estarmos lidando com previsões que podem fazer a diferença entre tratamentos mais indicados ao câncer em seres humanos e segundo pois apresenta instabilidade em performance.

Isso pode ser explicado pois o KNN precisa de muito mais dados de treinamento para estimar a função de mapeamento oferecendo assim melhor desempenho.

8.5.2. Palavras mais importantes de acordo com o modelo preditivo

Aqui temos uma vantagem deste modelo, podemos identificar como cada palavra impacta na previsão das classes individualmente.

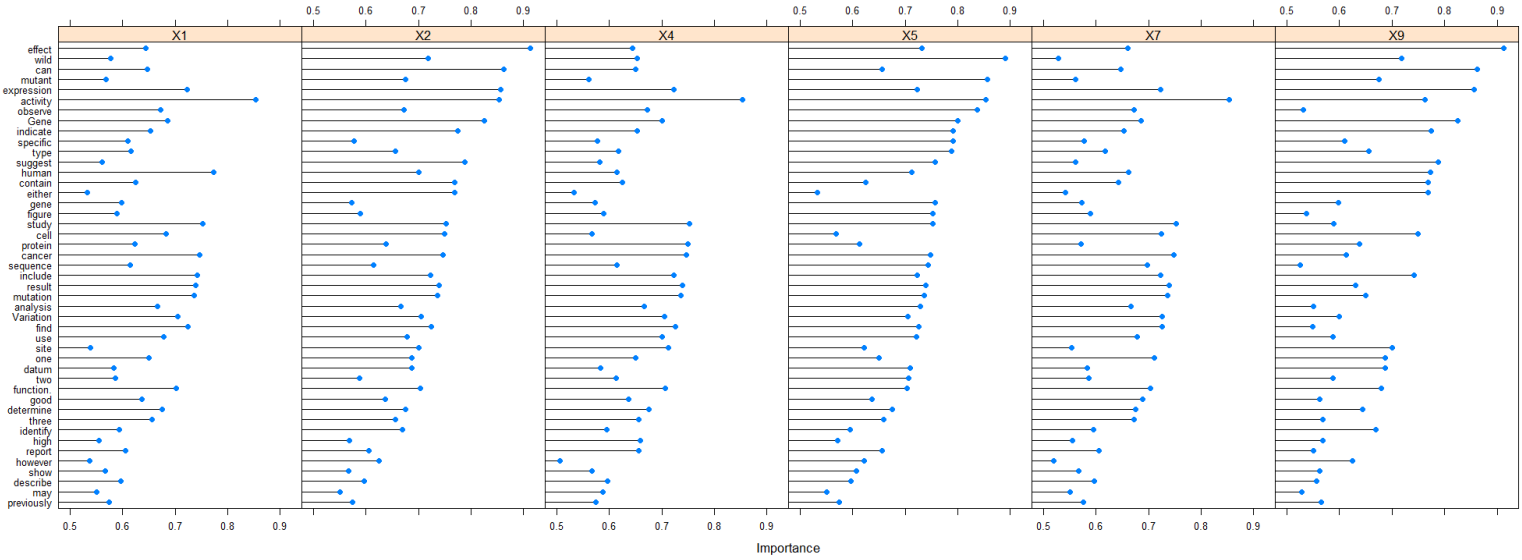

Figura 23 - Palavras Mais Importantes (KNN)

Esta análise é muito valiosa para estimar com mais precisão como cada classe se comporta nas evidências clínicas notadas pelos especialistas. Veja que a palavras activity é a mais relevante para o modelo nas classes 1, 4 e 7; nas classes 2 e 9 a palavra effect se destaca; e na classe 5 a palavra wild.

Realizei a aplicação da média nas palavras de cada classe e juntei os resultados obtidos encontrando assim as palavras mais relevantes presentes nas evidências clínicas para prever as classes apresentadas:

- activity, functional, method, demonstrate, expression, loss, Gene, effect, single

Conforme mencionei no capítulo onde analisamos Decision Tree, nosso próximo modelo será o RandomForest, uma união de várias Árvores de Decisões observando melhora nos resultados obtidos até o momento.

8.6 Random Forest

Os dois últimos modelos que vamos explorar serão modelos avançados e que oferecem melhores performances porém precisam de mais cuidado para ajuste dos parâmetros como é o caso do Random Forest pois um descuido pode levar à overfitting tornando um modelo poderoso em um modelo sem utilidade.

Com as variáveis selecionadas podemos treinar o modelo preditivo, porém um ponto fundamental é tentar identificar quando o modelo entra na zona de underfitting, quando encontra o menor erro (valor ideal) e quando chega na zona de overfitting. Antes vimos alguns conceitos sobre Árvore de Decisão (DecisionTree) e agora vamos estender estes conceitos ao Random Forest.

Como o nome sugere, randomForest significa Floresta Aleatória, o que em Data Science podemos fazer analogia à 2 ou mais Árvores de Decisões onde cada árvore possui uma profundidade e decide entre suas 'folhas' qual o melhor caminho a ser percorrido.

No randomForest crescemos múltiplas árvores ao invés de uma única árvore. Mas como funciona o processo de classificação? Inicialmente para classificar um novo objeto baseado em atributos, uma árvore gera uma classificação para esse objeto (que é como se a árvore desse votos para essa classe). Esse processo vai acontecendo para cada árvore presente na floresta e por fim, a floresta escolhe a classificação que tiver mais votos de todas as árvores da floresta. No caso de regressão, é considerado a média das saídas por árvores diferentes.

Para ilustrar o processo executado pelo randomForest, segue figura abaixo:

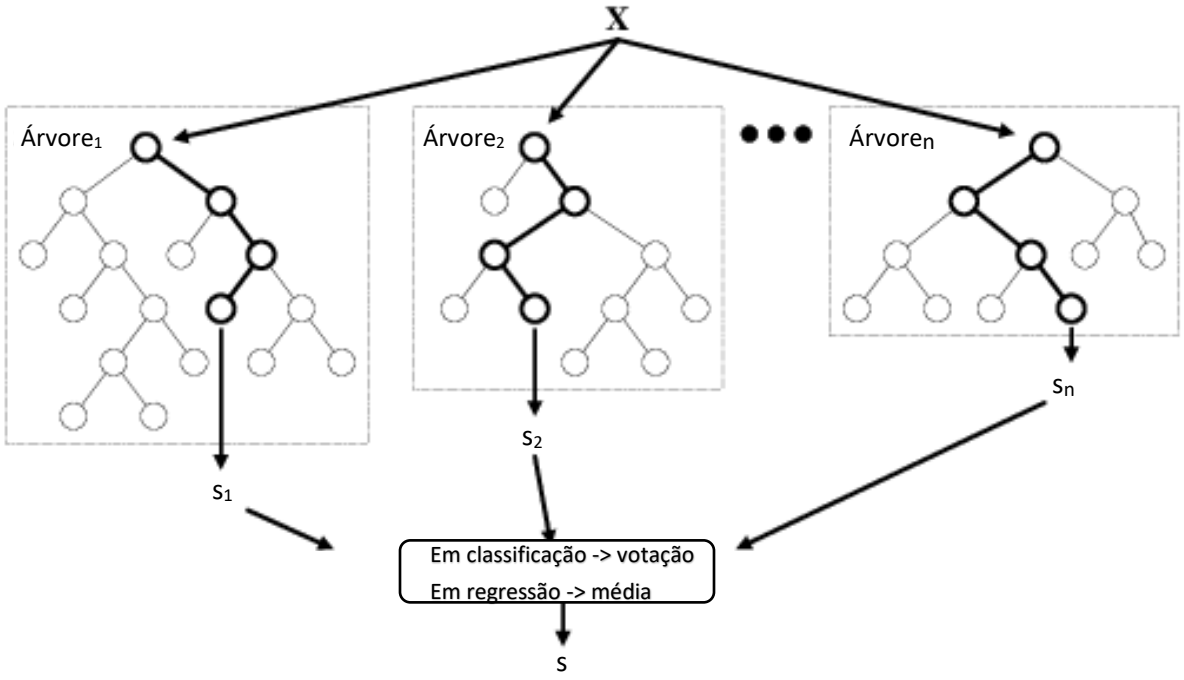

Figura 24 - Random Forest

Conforme figura acima, podemos ter n árvores e cada árvore pode ter quantas folhas desejar. É nesse momento que temos um problema, pois uma árvore rasa que foi treinada para classificar um objeto pode não oferecer precisão pois aprendeu pouco, ou em outras palavras, teve um *underfitting*. No outro extremo temos o *overfitting* (sobreajuste, ou super treinamento),

ou seja, se não for estabelecido um limite o modelo vai dar 100% de precisão no conjunto de treinamento pois ele acaba fazendo uma folha para cada observação.

Imagino que a pergunta agora seria: “Mas oferecer 100% de precisão não é bom? ”. A resposta é sim e não, pois é bom termos precisão, porém deste modo a precisão está apenas nos dados de treino e o meu objetivo é gerar um modelo de aprendizado de máquina imparcial onde qualquer dado possa ser previsto. No caso do *overfitting* quando eu apresentar novos dados (que são os dados de teste) o modelo vai falhar e retornar precisão insatisfatória.

Segue imagem ilustrando a problemática:

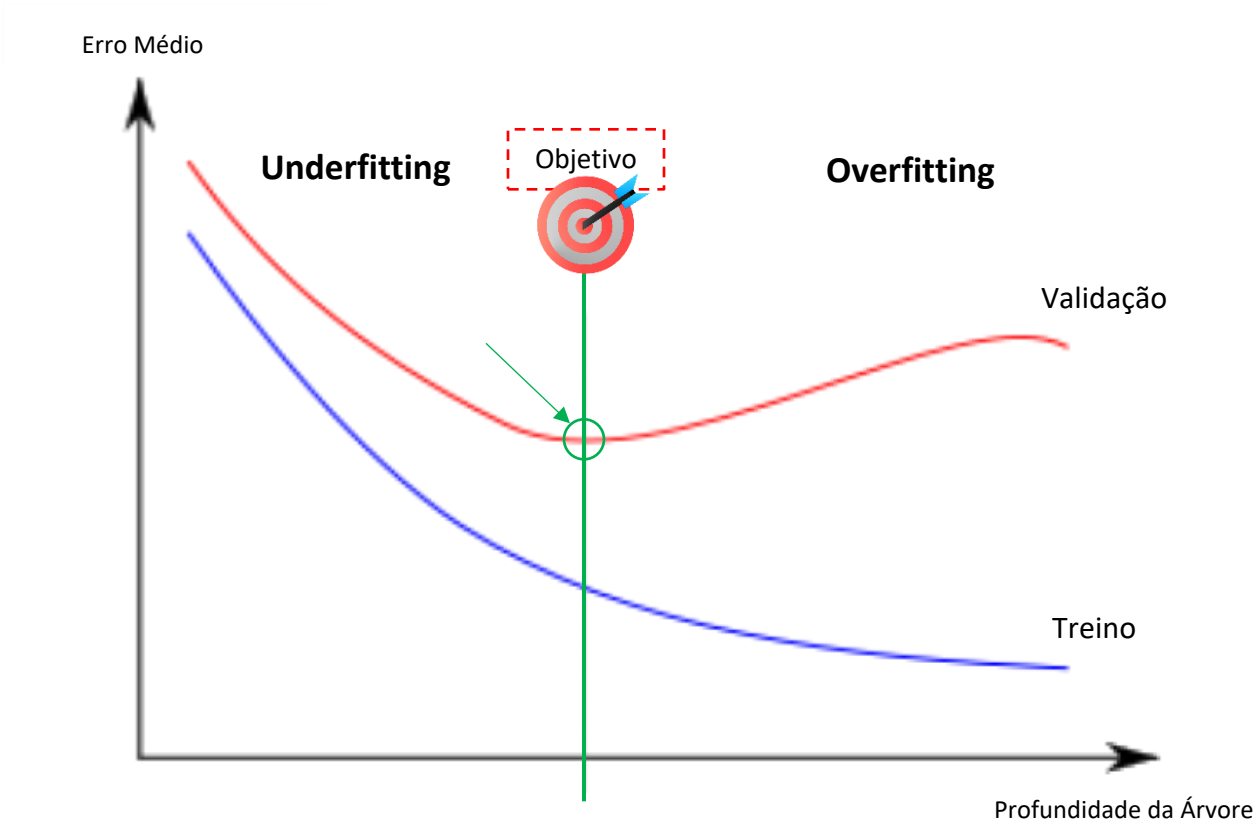

Figura 25 - Underfitting x Overfitting

Na figura acima temos 3 linhas com cores diferentes e cada uma representa uma informação importante:

- Linha Azul: representa o erro médio que os dados de treino fornecem de acordo com a profundidade da árvore. Quanto mais profundo, menor o erro visto que os dados de treino foram aprendidos quase que na totalidade.
- Linha Vermelha: representa o erro nos dados de teste (validação). Observe que o erro começa alto, diminui e em seguida aumenta novamente. Esse é um dos principais

desafios enfrentados ao modelar árvores de decisão, encontrar o ponto ideal em que o erro seja o menor possível.

- Linha Verde: conforme é possível observar, a linha verde cruza exatamente no ponto ideal, onde o erro nos dados de teste (validação) seja o mínimo possível oferecendo assim melhor precisão ao modelo.

Consolidando na figura a seguir o objetivo em realizar modelagem preditiva controlando underfitting e overfitting:

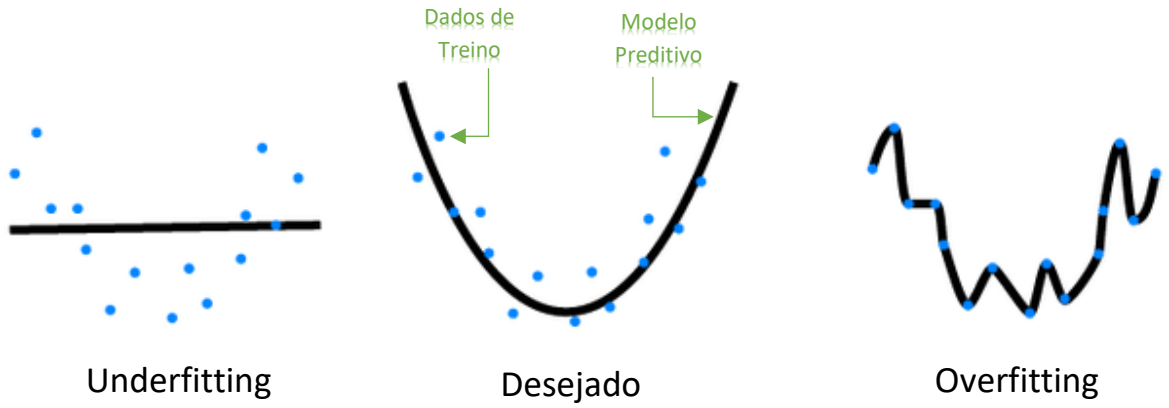

Figura 26 - Underfitting x Ideal x Overfitting

Conforme figura acima, o que desejamos é que o modelo tenha uma curva ideal evitando pouca precisão, mas que também não memorize 100% dos dados de treino falhando em novos e desconhecidos dados. Vejamos como o nosso modelo se comportou com os dados apresentados.

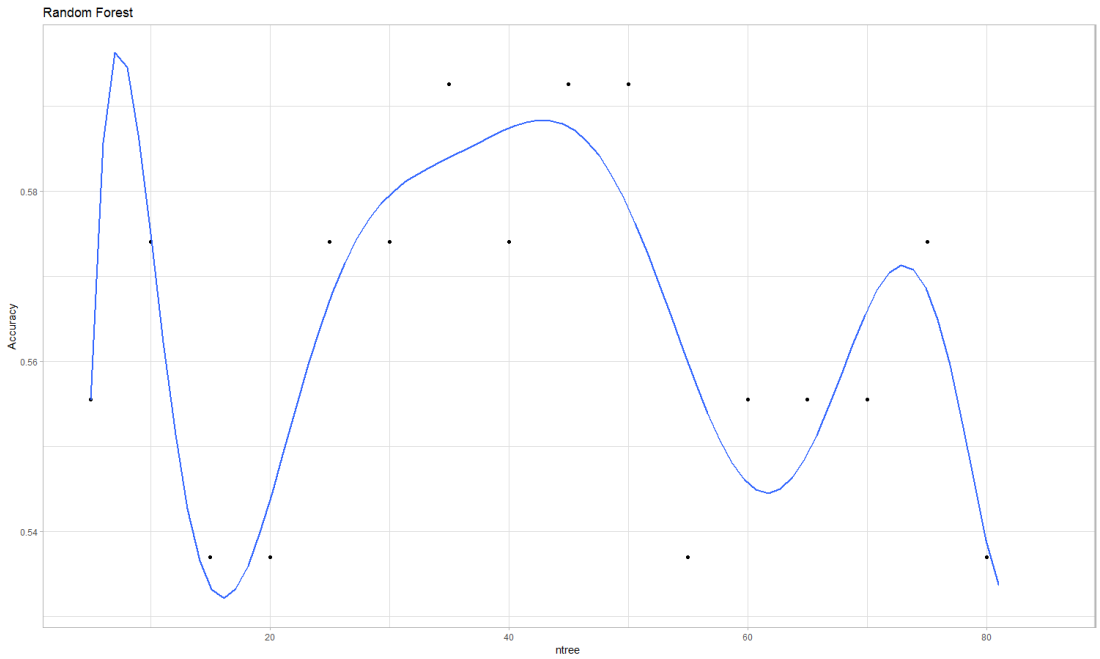

Figura 27 - Curva de Acurácia do modelo RandomForest

A figura 27 apresenta a variação na acurácia do modelo de modo que podemos observar estados de underfitting antes de 20 Árvores de Decisão com baixa precisão e estados de overfitting após 50 Árvores de Decisão também atuando precariamente na acurácia do modelo. Vamos utilizar o valor de 35 Árvores de Decisão como o ideal para a nossa base de conhecimento.

8.6.1. Confusion Matrix e Acurácia

Segue confusion matrix após apresentados dados de teste ao modelo Random Forest:

```
> confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')
Confusion Matrix and Statistics

    reference
data  1  2  4  5  7  9
   1 10  0  3  0  0  1
   2  0  1  0  0  3  0
   4  5  0 18  2  0  2
   5  0  0  0  0  0  0
   7  0  0  0  0  3  0
   9  2  0  0  0  0  4

Overall Statistics

               Accuracy : 0.6667
                 95% CI : (0.5253, 0.7891)
    No Information Rate : 0.3889
    P-Value [Acc > NIR] : 3.349e-05

                  Kappa : 0.5252

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 4 Class: 5 Class: 7 Class: 9
Sensitivity            0.5882  1.00000   0.8571  0.00000  0.50000  0.57143
Specificity            0.8919  0.94340   0.7273  1.00000  1.00000  0.95745
Pos Pred Value         0.7143  0.25000   0.6667      NaN  1.00000  0.66667
Neg Pred Value         0.8250  1.00000   0.8889  0.96296  0.94118  0.93750
Prevalence             0.3148  0.01852   0.3889  0.03704  0.11111  0.12963
Detection Rate         0.1852  0.01852   0.3333  0.00000  0.05556  0.07407
Detection Prevalence   0.2593  0.07407   0.5000  0.00000  0.05556  0.11111
Balanced Accuracy      0.7401  0.97170   0.7922  0.50000  0.75000  0.76444
>
```

Figura 28 - Confusion Matrix Random Forest

→ Métricas Nº 1

No modelo Random Forest temos resultados muito melhores quando consideramos 35 estimadores de Árvore de Decisão ao invés de apenas um. Observe que todas as classes (excetuando a classe 5) tiveram melhor performance na relação entre previsto e esperado. Inclusive classe majoritária 4 errou apenas 3 previsões. Melhoria na performance.

→ Métricas Nº 2

A acurácia do modelo quando apresentados dados desconhecidos:

Acurácia (Random Forest) -> 66.67%

Conforme vimos na métrica nº 1 a maior parte das classes foram corretamente classificadas, e a acurácia vem corroborar com mais um indicativo de boa performance chegando ao máximo de 66.67% de precisão.

→ Métricas Nº 3

Neste item temos métricas estatísticas indicando como cada classe se comporta em determinada análise. Podemos destacar:

- Balanced Accuracy: todas as classes (excetuando a classe 5) tiveram bons valores
- Specificity: aqui a média foi de 90% determinando que o nosso modelo está apto para previsões corretas.

É possível melhorar ainda mais o modelo Random Forest caso tivéssemos mais dados para alimentar o modelo, porém esta análise será feita ao final desta documentação, antes vamos apresentar as palavras mais importantes segundo o modelo.

8.6.2. Palavras mais importantes de acordo com o modelo preditivo

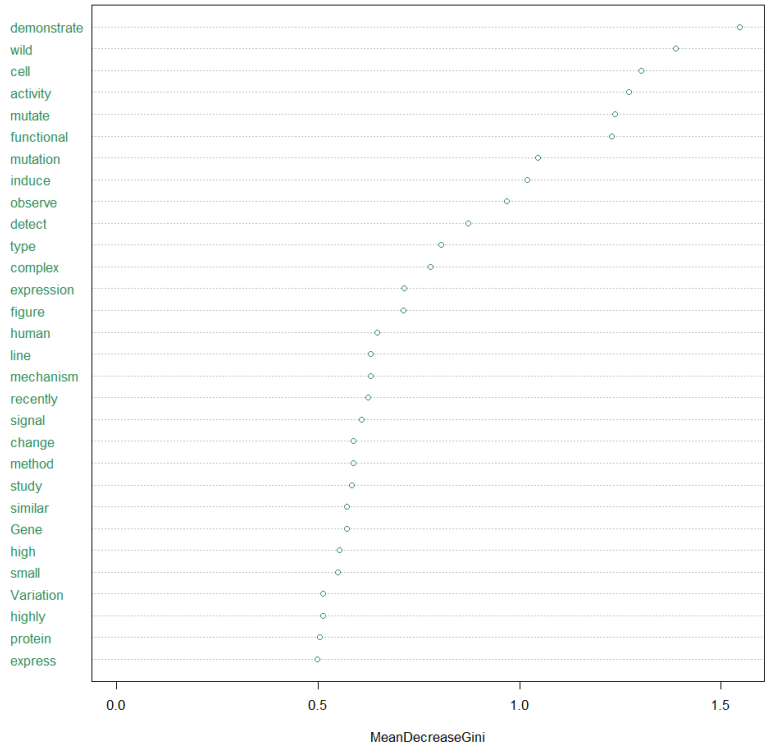

Figura 29 - Palavras Mais Importantes (KNN)

- demonstrate, wild, cell, activity, mutate, functional, mutation, induce, observe

8.7 Extreme Gradient Boosting (XGBoost)

Gradient Boosting ou Gradient Boostted Regression Trees (GBRT) é uma técnica de aprendizagem estatística não-paramétrica usada para problemas de "Classificação e Regressão", onde o mesmo cria Árvores de Decisão como estimadores. Ele é na verdade a junção de duas técnicas -> Gradient Boosting = Gradient Descent(descendente) + Boosting.

A adição do “extreme” ao nome se deve ao fato de levar o algoritmo ao limite dos recursos computacionais quando usados árvores de decisão, tornando ele rápido e preciso. Este algoritmo executa paralelização na construção de árvores usando todos os cores da CPU durante o treinamento; aplica conceitos de computação distribuída para o caso de grandes conjuntos de dados de treino; além de uso de cache a fim de otimizar a memória fazendo bom uso do hardware.

O que torna este algoritmo de Machine Learning tão poderoso é o uso dos conceitos de Boosting, ou seja, Boosting é uma técnica onde novos modelos são adicionados para corrigir erros de modelos já existentes, com isso o algoritmo adiciona sequencialmente novas árvores de decisão até que não se alcance mais melhorias no erro residual.

Junto ao Boosting temos o Gradient Descent tornando o modelo ainda mais poderoso nas previsões pois adota uma abordagem onde a adição de novos modelos além de olhar para o erro do modelo anterior ainda realiza a previsão do próximo erro residual e utiliza as duas informações para adicionar os novos modelo de Árvore de Decisão mais otimizado minimizando a função de perda.

Já executamos Árvore de Decisão individual e não obtivemos boa performance, mas juntamos mais do que uma Árvore de Decisão (35 pra ser exato) e construímos um RandomForest com notável melhoria na performance do modelo. Vamos agora então otimizar nossas Árvores de Decisão com o XGBoost e analisar como este modelo pode aprimorar nossos resultados.

Diferente do Random Forest aqui não preciso ficar analisando graficamente, ou por meio de tabelas de resultados, se o modelo entrou em zonas de overfitting pois já existem parâmetros a serem configurados que fazem com que o modelo execute esta verificação e minimize a chance de entrar na zona de overfitting.

Alguns dos parâmetros a serem configurados dentro do modelo são:

- eta: aceita valores entre 0 e 1, é um dos parâmetros que controla a taxa de aprendizagem. Um valor baixo para eta torna o modelo mais robusto e conservador para overfitting, porém em troca seus calculos se tornam mais lento;

- max_depth: profundidade máxima de cada Árvore de Decisão;
- gamma: redução de perda mínima necessária para fazer uma partição adicional em um nó folha da árvore. Quanto maior, mais conservador se torna o algoritmo;
- nthread: este é o número de paralelização usado para executar o modelo;
- early_stopping_rounds: quantidade de rounds em que a validação dos dados de treino no modelo vai utilizar como parâmetro de parada caso não haja melhorias.

O grande desafio deste modelo é saber ajustar os parâmetros de acordo com a sua capacidade computacional, conjunto de dados, tempo disponível para treino, melhor redução de perda entre outras especificações.

Através da visualização a seguir podemos ver as 5 primeiras Árvores do modelo com os pesos que cada elemento carrega até as folhas apresentando assim o ganho de informação obtido por cada variável.

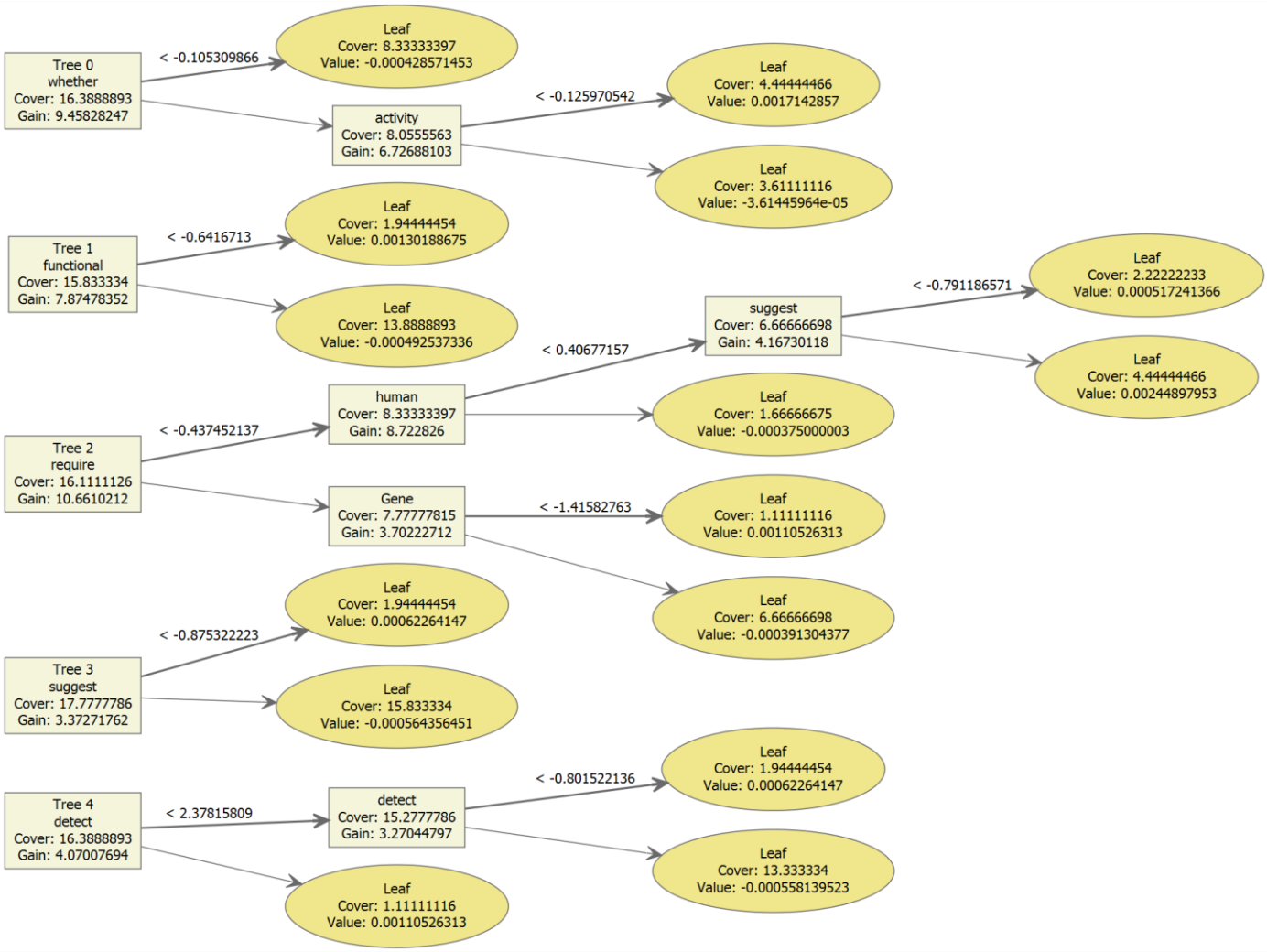

Figura 30 - XGBoost Tree

8.7.1. Confusion Matrix e Acurácia

Segue confusion matrix após apresentados dados de teste ao modelo XGBoost:

```
> confusionMatrix(table(data = predicted.classes$prediction, reference = predicted.classes$target))
Confusion Matrix and Statistics

    reference
data  1  2  4  7  9
   1 13  0  2  0  0
   2  0  1  3  1  1
   4  3  0 15  3  0
   7  0  0  0  2  0
   9  1  0  1  0  6

Overall Statistics

               Accuracy : 0.7115
                 95% CI : (0.5692, 0.8287)
    No Information Rate : 0.4038
    P-Value [Acc > NIR] : 6.953e-06

                  Kappa : 0.5967

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 4 Class: 7 Class: 9
Sensitivity            0.7647  1.00000   0.7143  0.33333   0.8571
Specificity            0.9429  0.90196   0.8065  1.00000   0.9556
Pos Pred Value         0.8667  0.16667   0.7143  1.00000   0.7500
Neg Pred Value         0.8919  1.00000   0.8065  0.92000   0.9773
Prevalence             0.3269  0.01923   0.4038  0.11538   0.1346
Detection Rate         0.2500  0.01923   0.2885  0.03846   0.1154
Detection Prevalence   0.2885  0.11538   0.4038  0.03846   0.1538
Balanced Accuracy      0.8538  0.95098   0.7604  0.66667   0.9063
> |
```

Figura 31 - Confusion Matrix XGBoost

Através das métricas na figura anterior já é possível notar a melhoria que este modelo trouxe para a precisão do modelo.

→ Métricas Nº 1

No modelo XGBoost temos resultados muito melhores quando comparados à todos até o momento. Observe que todas as classes (excetuando a classe 7) tiveram performances excelentes na relação entre previsto e esperado. Inclusive classe majoritária 1 errou apenas 4 previsões de 17. Incrível o poder deste algoritmo.

→ Métricas Nº 2

A acurácia do modelo quando apresentados dados desconhecidos:

Acurácia (XGBoost) -> 71.15%

Saímos de incríveis 46.30% de precisão da Árvore de Decisão para 71.15% com o modelo XGBoost utilizando as mesmas Árvores de Decisão, porém com métricas de ajuste e previsão de erros corrigindo os erros anteriores com o objetivo de minimizar a função de perda.

Conforme vimos na métrica nº 1 a maior parte das classes foram corretamente classificadas, e a acurácia vem corroborar com mais um indicativo de excelente performance chegando ao máximo de 71.15% de precisão, a melhor dentre todas anteriores.

→ Métricas Nº 3

Neste item de métricas estatísticas podemos destacar:

- Balanced Accuracy: excetuando a classe 7, todas tiveram ótimos resultados.
- Specificity: esta métrica que determina o quão apto o modelo está para realizar previsões corretas demonstrou que 90% das classes possuem mais do que 80% de valores ótimos para previsões. Isso mostra como Árvores de Decisão são simples porém poderosos estimadores quando unidos em algoritmos de alta performance.

Vejamos quais as palavras mais relevantes para este modelo.

8.7.2. Palavras mais importantes de acordo com o modelo preditivo

Através de um objeto de Cross – Validation utilizaremos todas as combinações possíveis entre variáveis realizando um processo iterativo com o modelo de Machine Learning resultando nas variáveis mais importantes para esse modelo.

De acordo com o modelo, as palavras mais relevantes presentes nas evidências clínicas para prever as classes apresentadas são:

- functional, mutate, human, activity, require, express, suggest, observe, Gene

9 – Conclusões

Todo o projeto demorou em média 1 minuto e 48 segundos para ser executado (considerando todos os modelos preditivos treinados e realizando previsões).

Para concluir a apresentação deste projeto vamos consolidar qual foi o modelo com melhor performance, quais as palavras mais importantes, menos importantes, e como está a distribuição de letras entre as palavras.

9.1 Apuração Modelos Preditivos

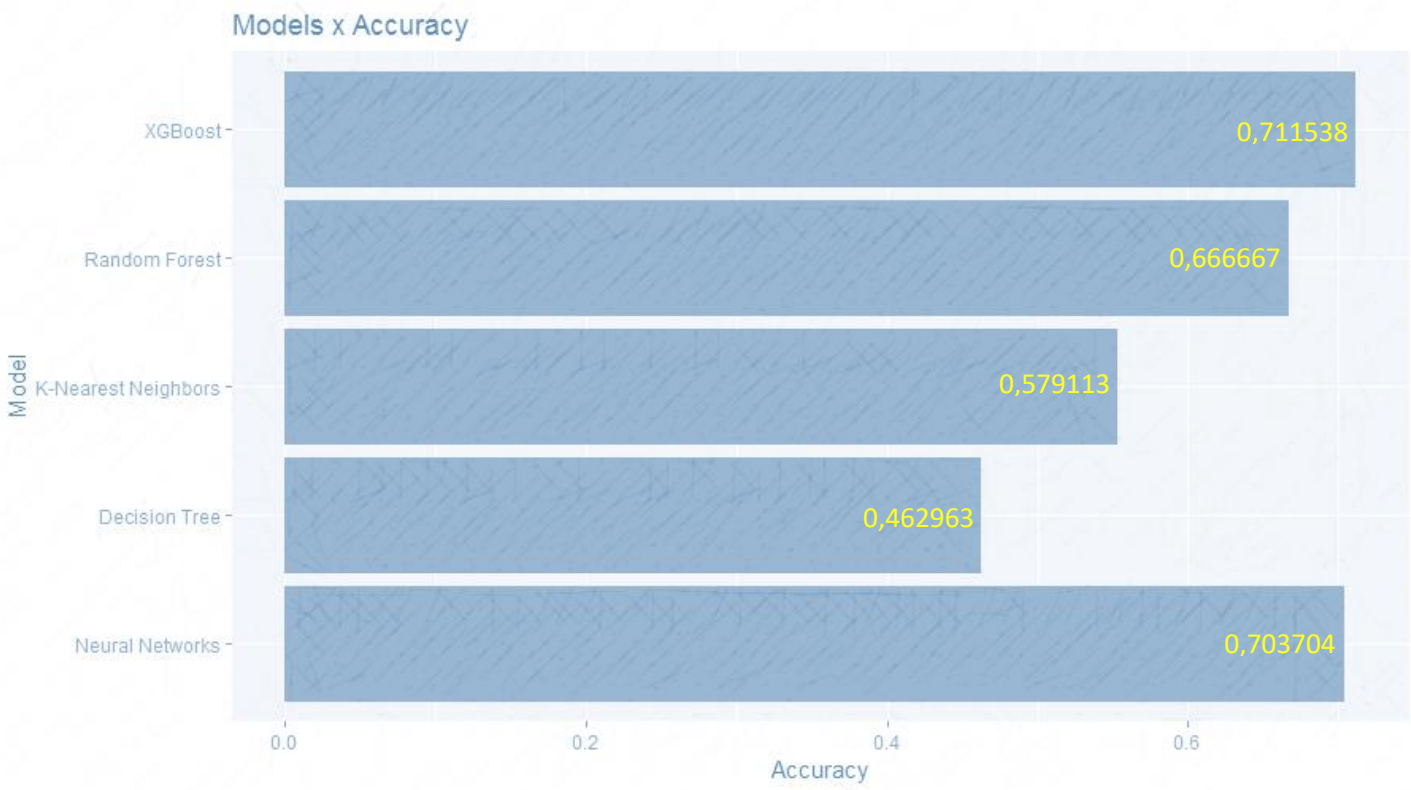

Figura 32 - Modelos x Acurácia

Conforme exposto nas análises individuais de cada modelo e confirmando acima, o XGBoost foi o campeão no quesito precisão em dados desconhecidos. Apesar de ser um modelo mais difícil de se ajustar devido à alta quantidade de parâmetros para serem configurados, ele certamente apresenta excelente performance quando comparados aos outros modelos.

Uma observação importante pode ser feita quanto ao modelo de Redes Neurais Artificias que na sua "simplicidade" ofereceu quase o mesmo ganho de qualidade em previsões. Um algoritmo poderoso e que pode oferecer também bons resultados mesmo sendo uma espécie de caixa preta nas suas camadas intermediárias e ocultas.

9.2 Consolidação das Palavras Mais Importantes

Cada modelo preditivo realizou a sua análise e considerou na base de conhecimento as palavras mais importantes a seguir para prever cada classe:

Figura 33 - Palavras Mais Importantes (apuradas pelos modelos preditivos)

9.3 Consolidação das Palavras Menos Importantes

Considerou também as seguintes palavras como não tão relevantes assim para o processo de previsão:

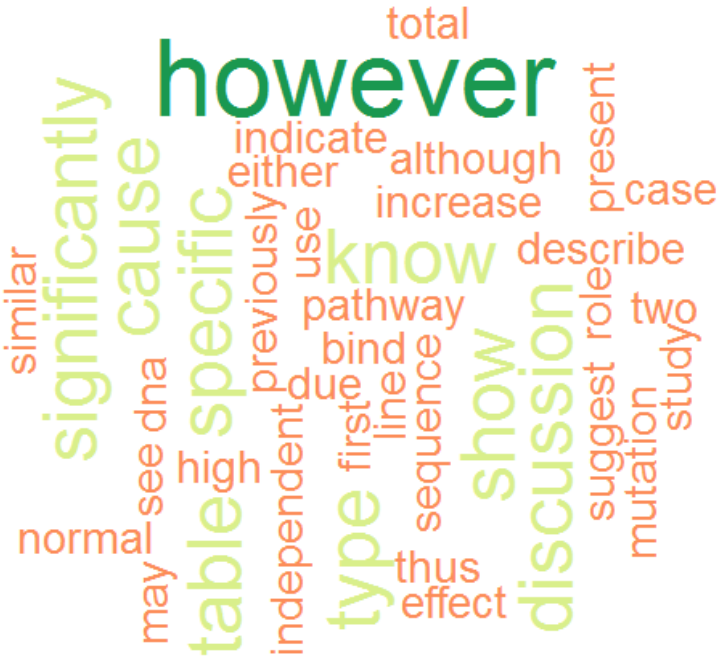

Figura 34 - Palavras Menos Importantes (apuradas pelos modelos preditivos)

9.4 Letras por Palavras Mais Relevantes

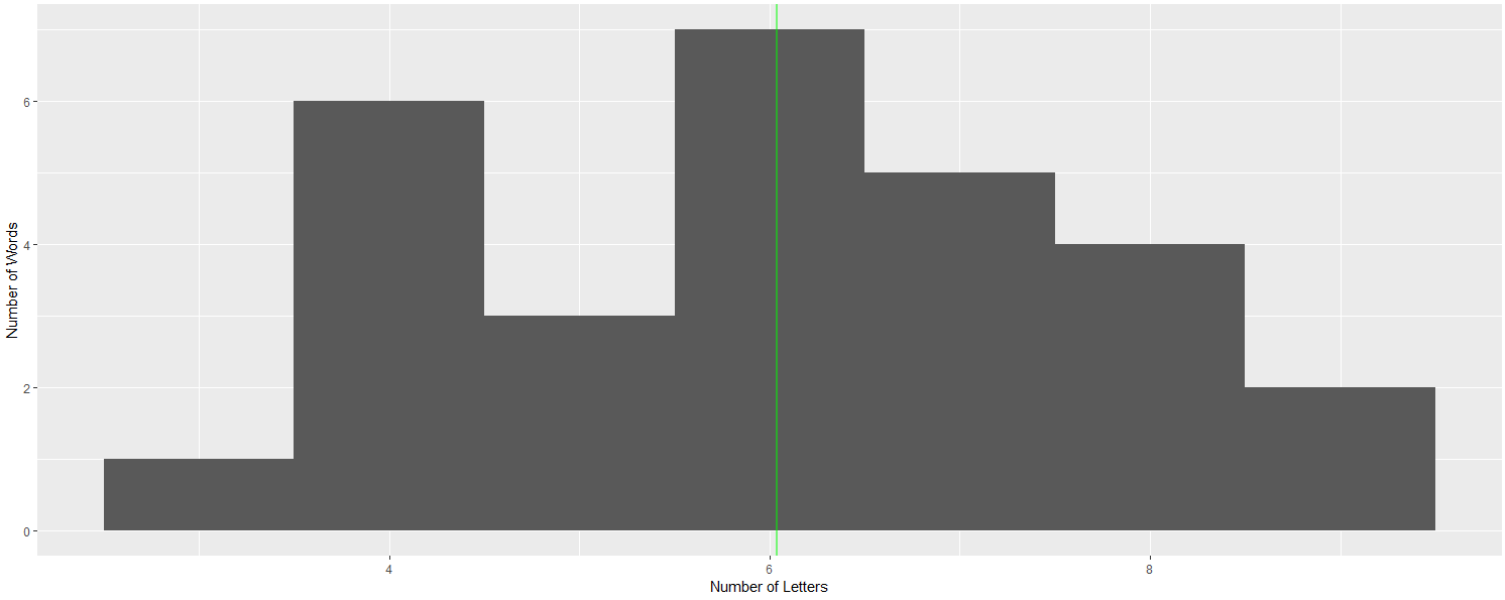

Figura 35 - Letras por Palavras Mais Relevantes (apuradas pelos modelos preditivos)

Observe que os modelos preditivos consideram que em média as palavras mais relevantes possuem até 6 letras caindo a relevância das palavras conforme as letras vão aumentando nas palavras.

9.5 Letras por Palavras Menos Relevantes

O oposto ocorre quando olhamos para as palavras que não são consideradas boas para previsões. Observe no gráfico a seguir como o aumento de letras descaracteriza boas variáveis, chegando ao pico quando temos oito letras por palavra.

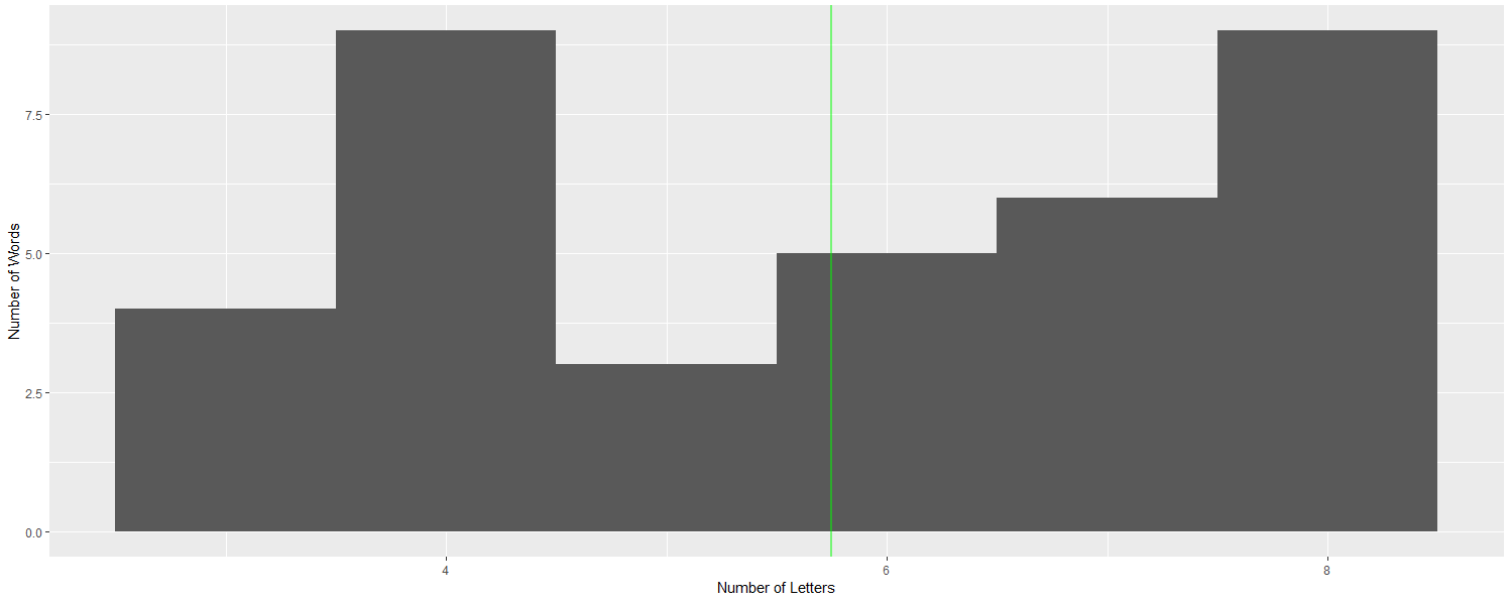

Figura 36 - Letras por Palavras Menos Relevantes (apuradas pelos modelos preditivos)

9.6 Considerações Finais

Certamente a fase mais crítica e decisiva de todo o projeto foi o pré-processamento da base de conhecimento onde temos a oportunidade de moldar os dados e eliminar ou valorizar variáveis conforme análise exploratória.

A maior parte de tempo gasto foi justamente neste processo de limpeza e transformação onde as palavras tiveram que ser ajustadas e modeladas para que fosse possível um melhor entendimento do problema de negócio.

Durante a análise quantitativa foi possível realizar uma série de insights como palavras mais frequentes, menos frequentes, esparsidade correlações e interligações entre palavras. Este processo evidenciou que podemos estar tratando de uma base de conhecimento relacionada ao câncer de mama e estudos correlacionados em diferentes locais do mundo. Um ponto interessante para melhoria do projeto neste quesito seria unir especialistas patológicos com Cientistas de Dados e juntos analisarem quais palavras podem se tornar ótimas variáveis para se apresentar à modelos preditivos.

Como aqui estamos apenas realizando testes em base de conhecimento já anotadas e não temos acesso à especialistas, assumi uma abordagem conservadora e decidi diminuir a esparsidade da nossa base de dados de 93% para 20% onde termos menos frequentes deixaram de fazer parte das previsões. Um dos maiores motivos para essa abordagem se deve ao fato de que muitos dados, algo em torno de 5 milhões de variáveis, dificilmente fariam os modelos de Machine Learning convergirem em resultados satisfatórios.

Ao apresentar os dados de uma matriz menos esparsa aos modelos preditivos conseguimos fazer com que todos os 5 modelos convergissem em seus resultados finais sendo que o mais adequado foi o XGBoost trazendo aproximadamente 72% de precisão quando novos e desconhecidos dados são apresentados ao modelo, conforme explorado na confusion matrix.

Além de prever novas classes, os modelos preditivos auxiliaram em outras funcionalidades como apresentação das variáveis mais importantes (e não as mais frequentes) para o processo de previsão, evidenciando também qual a quantidade de caracteres que uma palavra menos importante possui. Em nosso caso, palavras com muitas letras não ajudam a prever as classes corretamente, e em média palavras com 6 caracteres se mostraram mais adequadas para melhorias em acurácia.

Como estamos lidando com vidas humanas, 72% está longe de ser um resultado ideal para colocar o modelo em produção, sendo assim, para melhoria do projeto acredito que o ideal seria coletar mais dados e explorar, junto com especialistas da área, qual a melhor configuração de variáveis para maximizar o ganho em acurácia.

Código Fonte

```
# Personalized Medicine: Redefining Cancer Treatment: Predict the effect of Genetic Variants to enable Personalized Medicine ----
  ----------

###  Defining the Business Problem ###

# There are NINE different CLASSES a genetic mutation can be classified on.
# Later on, we will notice a problem with 3 classes and how to deal with them.

# This is not a trivial task since interpreting clinical evidence is very challenging even for human specialists.
# Therefore, modeling the clinical evidence (text) will be critical for the success of your approach.

# Both, training and test, datasets are provided via two different files.
# One (training/test_variants) provides the information about the genetic mutations, whereas
# the other (training/test_text) provides the clinical evidence (text) that our human experts used to classify the genetic mutations.
# Both are linked via the ID field.

# Therefore the genetic mutation (row) with ID=15 in the file training_variants, was classified using the clinical
# evidence (text) from the row with ID=15 in the file training_text

## Data Dictionary ------------------------------------------------------------------------------
# File descriptions:
#  training_variants - a comma separated file containing the description of the genetic mutations used for training.
#               -> ID (the id of the row used to link the mutation to the clinical evidence),
#               -> Gene (the gene where this genetic mutation is located),
#               -> Variation (the aminoacid change for this mutations),
#               -> Class (1-9 the class this genetic mutation has been classified on) -> TARGET

#  training_text - a double pipe(||)delimited file that contains the clinical evidence (text) used to classify genetic mutations.
#               -> ID (the id of the row used to link the clinical evidence to the genetic mutation),
#               -> Text (the clinical evidence used to classify the genetic mutation)

#  test_variants - a comma separated file containing the description of the genetic mutations used for training.
#               -> ID (the id of the row used to link the mutation to the clinical evidence),
#               -> Gene (the gene where this genetic mutation is located),
#               -> Variation (the aminoacid change for this mutations)

#  test_text - a double pipe(||)delimited file that contains the clinical evidence (text) used to classify genetic mutations.
#               -> ID (the id of the row used to link the clinical evidence to the genetic mutation),
#               -> Text (the clinical evidence used to classify the genetic mutation)

## Directory ---------------------------------------------------------------------------------------------------
# Configuring the working directory
# Quote the working directory you are using on your computer.

# SET WORKING AND RESULTS DIRECTORY
working_directory <- "C:/FCD/4MachineLearning/Cap22/R1"
results_directory <- "C:/FCD/4MachineLearning/Cap22/R1/Results7"
```

```
setwd(working_directory)
# GETTING CURRENT DIRECTORY
getwd()

## Library ----------------------------------------------------------------------------
#install.packages("tm")
#install.packages("textstem")
#install.packages("qdap")
#install.packages("qdapDictionaries")
#install.packages("dplyr")
#install.packages("RColorBrewer")
#install.packages("ggplot2")
#install.packages("scales")
#install.packages("ROCR")
#install.packages("caret")
#install.packages("e1071")

# Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. Bioconductor uses the
# R statistical programming language, and is open source and open development. It has two releases each year, and an
# active user community.
# http://bioconductor.org/
# The command below will not work because it is from 2017 and the Bioconductor has been updated
source("http://bioconductor.org/biocLite.R")    # Biocondutor
# This command is the updated one (2020) according to the Bioconductor website
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install(version = "3.10")

BiocManager::install("Rgraphviz")
#biocLite("Rgraphviz")                    # Correlation plots between words

# Loading Packages
# Package to use "read_delim" so I'll read a dataset with "\t" separator
library(readr)
# Framework for text mining
library(tm)
# Framework for text mining, to use especifically the "lemmatization_strings()" function
library(textstem)
# Data Manipulation
library(dplyr)
# Correlation Plots between words, Bioconductor's function
library(Rgraphviz)
# Graphic Package
library(ggplot2)
# Color Palette
library(RColorBrewer)
# Dictionary
library(qdapDictionaries)
# Quantitative discourse analysis of trasncripts
library(qdap)
# Allows you to include commas in numbers, creating thousands
```

```
library(scales)
# Package with some Machine Learning Models
library(e1071)
# Package to use 'confusionMatrix' function
library(caret)
# Package to assist in the construction of evaluation metrics, such as confusion matrix for example.
library(ROCR)

# Viewing the data sources (Function of the tm package)
getSources()

# Readers (tm package function)
getReaders()

## Datasets --------------------------------------------------------------------------------
time_init <- Sys.time()
# Dataset with the description of the genetic mutations
dfTrain_variants <- read.csv("training_variants")
# Verifying if there's NA in the dataset: NO, there is not NA
sum(is.na(dfTrain_variants))

# Dataset with the clinical evidence (text) used to classify genetic mutations.
# This dataset gave so many problems because the separator is double pipe ("||") and R doesn't accept more than one separator,
# so I first parsed with another software changing "||" for "tab" and then used readr package here
dfTrain_text <- read_delim("training_text", "\t", escape_double = FALSE, trim_ws = TRUE)
dfTrain_text <- as.data.frame(dfTrain_text)
str(dfTrain_text)
# Verifying if there's NA in the dataset: NO, there is not NA
sum(is.na(dfTrain_text))

# Merging Both Datasets by ID
df <- merge(dfTrain_variants, dfTrain_text, by = 'ID' )
str(df)
# As observed above, most columns are in the wrong type so I'll adjust it

# Adjusting column types
# Function to transform features into character
toChar <- function(var, dataset){
 for (i in var) {
  dataset[[i]] <- as.character(paste(df[[i]]))
 }
 return(dataset)
}

# Function to transform features into factors
toFact <- function(var, dataset){
 for (i in var) {
  dataset[[i]] <- as.factor(paste(df[[i]]))
 }
 return(dataset)
}
```

```
# Variables to Adjust
charVar <- c('Gene', 'Variation', 'Text')
df <- toChar(charVar, df)
factVar <- c('Class')
df <- toFact(factVar, df)
str(df)

# Loking at our final dataset
#View(df)

# Cleaning Memory
rm('charVar', 'factVar', 'dfTrain_variants', 'dfTrain_text', 'toChar', 'toFact')

# Preparing a Corpus ----------------------------------------------------------------------------------------------------------------------------
  ----
# To use the function "DataframeSource" I must transform my dataframe into some columns (doc_id, text, and metadatas)
# stringsAsFactors -> IMPORTANT parameter so my dataframe(df) stay as configured above
dfSource <- data.frame(doc_id = df$ID,
                       text = df$Text,
                       gene = df$Gene,
                       variation = df$Variation,
                       target = df$Class,
                       stringsAsFactors = FALSE)
str(dfSource)

# Creating a "DataframeSource" object, an object required by Corpus function
dfSource <- DataframeSource(dfSource)
# Creating a Corpus object
docs <- Corpus(dfSource)
#inspect(docs)
#meta(docs)

# Text Mining: Cleaning and Transforming ---------------------------------------------------------------------------------------------------------
  ----------

# After EACH cleaning or transformation, run both lines below to see if the transformation was made correctly
# Cleaning Console
cat("\014")
# Looking at the first text of Corpus object
docs[["0"]][["content"]]

# Possible transformations from 'tm' package
getTransformations()

# Changing what I think it's necessary (by a pattern I'll pass to a function) ------------------------------------------------------------------
# Creating a function to make some changes in the text
# content_transformer -> Create content transformers, i.e., functions which modify the content of an R object.
  ('content_transformer(FUN)')
# gsub -> search for a 'pattern' in 'x'(this 'x' will be my 'docs' object) and change it for another specified (" " in this case, i.e. change
  it to an space character)
```

```
toSpace <- content_transformer(function(x, pattern) gsub(pattern, " ", x))

# tm_map -> Interface to apply transformation functions (also denoted as mappings) to Corpus.
docs <- tm_map(docs, toSpace, "-")
docs <- tm_map(docs, toSpace, "/")
docs <- tm_map(docs, toSpace, "@")

#docs <- tm_map(docs, toSpace, "\\|")
#docs <- tm_map(docs, toSpace, "/|@|\\|")

# Making some transformation with Packages Functions ----------
# All letters to lower (using a function of the base system)
docs <- tm_map(docs, content_transformer(tolower))

# Now removing numbers with a getTransformation () function
docs <- tm_map(docs, removeNumbers)

# Removing Punctuation with a function of getTransformation()
docs <- tm_map(docs, removePunctuation)

# Removing stopwords ----------
# Stopwords in English
length(stopwords("english"))
stopwords("english")

# Here we are removing stopwords in english with the aid of a function from getTransformation()
docs <- tm_map(docs, removeWords, stopwords("english"))

# Here we are removing MINE stopwords that aren't in the stopwords ("english") list
docs <- tm_map(docs, removeWords, c("figur", "figur ", "fig", "also"))

# Removing blank spaces with the aid of a function from getTransformation()
docs <- tm_map(docs, stripWhitespace)

# Transforming Specific Texts ----------
# Function to change specific texts
# content_transformer -> Create content transformers, i.e., functions which modify the content of an R object.
# gsub -> search for a 'pattern' in 'x'(this 'x' will be my 'docs' object) and change it 'from' something 'to' another something
toString <- content_transformer(function(x, from, to) gsub(from, to, x))

# tm_map -> Interface to apply transformation functions (also denoted as mappings) to Corpus.
docs <- tm_map(docs, toString, "Gene", "gene")
#docs <- tm_map(docs, toString, "shenzhen institutes advanced technology", "SIAT")
#docs <- tm_map(docs, toString, "chinese academy sciences", "CAS")

# Lemmatization ----------
# Lemmatization adjust words to its Lemma
docs <- tm_map(docs, lemmatize_strings)

# Stemming ----------
# Stemming removes prefixes and suffixes. I'll use a function from getTransformation()
```

```
#docs <- tm_map(docs, stemDocument)
# I'VE REMOVED STEMMING BECAUSE IT JUST MESSED UP THE WORDS. LEMMATIZATION GAVE BETTER RESULTS.

# Encoding ----------
### CONVERTING THE ENCONDING OF THE DOCUMENT. WINDOWS PROBLEMS ###
docs <- tm_map(docs, enc2utf8)

# Quantitative Analysis ----------
# Converting Corpus 'docs' to perform a Quantitative Analysis
# As Machines don't understand strings (texts), we'll need to convert them to mathematical formulas, to do so let's build a DocumentTermMatrix that is a Matrix with
# frequency count of each term of the documents

# DocumentTermMatrix -> A Term Document Matrix is a matrix with Documents in the lines, Terms in the columns and a frequency count in the cells of the Matrix
dtm <- DocumentTermMatrix(docs)

# A DocumentTermMatrix is really too much sparse (i.e. mostly empty) and because of this it is stored in a better and compacted representation in the system
# Looking at the amount of lines(documents) and columns(terms)
dim(dtm)
# We have 138 lines and 15.986 columns(i.e. 15.986 words), that's correct according to the original dataset (training_text)
class(dtm)
# Inspecting some rows and columns of this matrix:
inspect(dtm[1:138, 20:30])
# Inspecting all Matrix
inspect(dtm)
# Look that "cancer" shows up 398 times in the document 96 and that Sparsity = 93% to this amount of data

# Let's look at the frequency of the terms in these docs:
# First I need to convert this DocumentTermMatrix to an Matrix
dtm_Matrix <- as.matrix(dtm)
# Second we'll count the sum of each column
freq <- colSums(dtm_Matrix)
length(freq)
# Ordering these frequency
# order -> ordering the frequency of the terms in ascending order. Note that this function gets the number of the columns, and not the name, so I'll need to perform another
#         code to retrieve the names and number of terms (i.e. "freq[head(ord)]" and "freq[tail(ord)]")
ord <- order(freq)
# Lowest terms
freq[head(ord)]
# Highest terms
freq[tail(ord,100)]
# NOTE THAT THE HIGHEST TERMS ARE mutation, cell, cbl, cancer. All related to this bussiness problem

# Preparing to plot "dtm - Most Frequent Words"
# Building a Dataframe with 30 Highest Terms
freq_df <- as.data.frame(freq[tail(ord, 30)])
# Renaming dataframe columns
```

```
colnames(freq_df) <- c('freq')
# Adding feature names of each frequency
freq_df$ feat <- rownames(freq_df)

ggplot(data = freq_df, aes(x = feat, y = freq)) +
  geom_bar(stat = "identity", fill='springgreen1') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("dtm - Most Frequent Words")
setwd(results_directory)
ggsave("1-dtm - Most Frequent Words.png", width = 10, height = 5)
setwd(working_directory)

# Preparing to plot "dtm - Least Frequent Words"
# Building a Dataframe with 30 Lowest Terms
freq_df <- as.data.frame(freq[head(ord, 30)])
# Renaming dataframe columns
colnames(freq_df) <- c('freq')
# Adding feature names of each frequency
freq_df$ feat <- rownames(freq_df)

ggplot(data = freq_df, aes(x = feat, y = freq)) +
  geom_bar(stat = "identity", fill='lightcoral') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("dtm - Least Frequent Words")
setwd(results_directory)
ggsave("2-dtm - Least Frequent Words.png", width = 10, height = 5)
setwd(working_directory)

# GOOD PRACTICES: Converting dtm matrix to a csv file and saving in my SO. In this way I'm preserving the job done until here.
setwd(results_directory)
m <- as.matrix(dtm)
dim(m)
write.csv(m, file="dtm.csv")
setwd(working_directory)

# Quantitative Analysis ----------------------------------------------------------------------------------------------
# Now I don't have a Corpus anymore, instead I have a Matrix in the format "Term Document" with frequency of Terms.
# Now, we'll perform some Quantitative Analysis to find relationship between words, most frequent words, what is the
#   relationship between words (i.e., identify whether two words appear together in the text and how often they appear
#   together, thus generating insights for the analysis process)
# Here it may be necessary to involve the business area beacuse, for example: let's suppose that this analysis is being
#   done for a law firm searching in a series of PDF documents. It may be that a Data Scientist does not know
#   about legal terms that would be relevant to this analisys.

# Removing Sparse Terms
# Sometimes we are not interested in infrequent terms in our documents, so, sparse terms might be removed from the
#   matrix term document easily by using removeSparseTerms() function
dim(dtm)
dtm
# removeSparseTerms() -> removing less frequent words
# dtm -> DocumentTermMatrix built previously
```

```
# 0.19-> A numeric for the maximal allowed sparsity in the range from bigger zero to smaller one, i.e., here I'm reducing
#       Sparsity from 93% to until 19%
dtms <- removeSparseTerms(dtm, 0.19)
dim(dtms)
dtms
# Look above that Sparsity reached 10% and we have gone from 15986 terms to 108
inspect(dtms)

# Terms remaining in the Matrix after reducing Sparcity:
freq <- colSums(as.matrix(dtms))
ord <- order(freq)
freq[head(ord, 20)]

# Preparing to plot "dtms - Most Frequent Words"
# Building a Dataframe with 30 Highest Terms
freq_df <- as.data.frame(freq[tail(ord, 30)])
# Renaming dataframe columns
colnames(freq_df) <- c('freq')
# Adding feature names of each frequency
freq_df$ feat <- rownames(freq_df)

ggplot(data = freq_df, aes(x = feat, y = freq)) +
  geom_bar(stat = "identity", fill='springgreen1') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("dtms - Most Frequent Words")
setwd(results_directory)
ggsave("3-dtms - Most Frequent Words.png", width = 10, height = 5)
setwd(working_directory)

# Preparing to plot "dtms - Least Frequent Words"
# Building a Dataframe with 30 Lowest Terms
freq_df <- as.data.frame(freq[head(ord, 30)])
# Renaming dataframe columns
colnames(freq_df) <- c('freq')
# Adding feature names of each frequency
freq_df$ feat <- rownames(freq_df)

ggplot(data = freq_df, aes(x = feat, y = freq)) +
  geom_bar(stat = "identity", fill='lightcoral') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("dtms - Least Frequent Words")
setwd(results_directory)
ggsave("4-dtms - Least Frequent Words.png", width = 10, height = 5)
setwd(working_directory)

# I WILL NOT USE dtms WITH THE REMOVAL OF THE SPARSE TERMS, IT WAS JUST A TRICK TO VIEW THE BEST WORDS #
# dtms WILL BE USED IN THE MACHINE LEARNING PROCESS

# Comparing frequent term in dtm and dtms
# findFreqTerms -> Package of "tm" that finds frequent terms in a document-term or term-document matrix.
findFreqTerms(dtm, lowfreq = 1000)   # Terms that appears 1000 times or more
```

```
findFreqTerms(dtms, lowfreq = 1000)  # Terms that appears 1000 times or more
findFreqTerms(dtm, lowfreq = 100)    # Terms that appears 100 times or more
findFreqTerms(dtms, lowfreq = 100)   # Terms that appears 100 times or more

# Searching for word associations and specifying the correlation limit
# If two words always appear together then the correlation would be 1.0 and if they never appear together
# the correlation would be 0.0. Thus, correlation is a measure of how words are together in the corpus.
# findAssocs -> Package of "tm" that finds associations in a document-term or term-document matrix.
# dtm -> document-term original
# "data" -> term to be located and correlated with others
# corlimit -> limit in which correlation must be analysed, i.e. from 1.0 to 0.66
findAssocs(dtm, "cancer", corlimit = 0.66)

# THIS IS AMAZING, WONDERFUL!!!!!!
# Analyzing graphically how is the relationship between the words that most appear together.
# Correlation plot using the Bioconductor RGraphViz package
plot(dtm, terms = findFreqTerms(dtm, lowfreq = 1000)[30:50], corThreshold = 0.5)
plot(dtm, terms = findFreqTerms(dtm, lowfreq = 100)[50:120], corThreshold = 0.5)

# The barploting below was moved to a before analysis (to line 368), here I'm just showing how to to the same in another way
# Barploting the frequency of the words
# Acquiring the Frequency of Words listing in descending order
freq <- sort(colSums(as.matrix(dtm)), decreasing = TRUE)
head(freq, 15)
# Acquiring the name of each word and its frequency, placing it in a data.frame to be able to plot with ggplotot
wf <- data.frame(word=names(freq), freq=freq)
head(wf)
# Ploting with ggplot2 frequencies above 2000
subset(wf, freq>2000) %>%
  ggplot(aes(word, freq)) +
  geom_bar(stat = "identity", fill='gold1') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("Words with > 2000 Frequencies")

# Wordcloud Visualization
#install.packages("wordcloud")
library(wordcloud)

# Getting at MAXIMUM 300 words
set.seed(666)
wordcloud(names(freq), freq, max.words = 300)

# Getting words which MINIMUM frequency is 300
set.seed(666)
wordcloud(names(freq), freq, min.freq = 300, colors = brewer.pal(6, "Dark2"))

# Getting words which MINIMUM frequency is 300 (BUT SCALING THE WORDCLOUD TO A SMALLER SHAPE)
set.seed(666)
wordcloud(names(freq), freq, min.freq = 300, scale=c(5, .1), colors = brewer.pal(6, "Dark2"))

# Getting words which MINIMUM frequency is 300 (BUT FORCING 50% OF PROPORTION WORDS WITH 90 DEGREE ROTATION)
```

```
setwd(results_directory)
png('7-WordCloud(Frequency).png', width = 700, height = 700, res = 100)
set.seed(666)
dark2 <- brewer.pal(6, "Dark2")
wordcloud(names(freq), freq, min.freq = 300, rot.per = 0.5, colors = dark2)
dev.off()
setwd(working_directory)

# Transforming dtm into a matrix, getting the names of the columns and searching for words with less than 10 characters
words <- dtm %>%
  as.matrix %>%
  colnames %>%
  (function(x) x[nchar(x) < 10])

# How many words are left?
length(words)
# Looking at 15 words
head(words, 15)
# Number of letters and how many words with that amount of letters
table(nchar(words))

# NOT WORKING!!! (because of some dependencies)
# dist_tab -> it shows a very nice table indicating the number of letters, the frequency and the percentage of the letters in the
   whole.
#dist_tab(nchar(words))

# Inserting "words" in a dataframe and plotting (basicaly I'm plotting freq column from dist_tab above)
setwd(results_directory)
png('8-Letters_x_Words.png', width = 1500, height = 600, res = 100)
data.frame(nletters=nchar(words)) %>%
  ggplot(aes(x=nletters)) +
  geom_histogram(binwidth=1) +
  geom_vline(xintercept=mean(nchar(words)),
             colour="green", size=1, alpha=.5) +
  labs(x="Number of Letters", y="Number of Words")
# What we have on the graph is on the x axis the number of letters and on the y axis the number of words, the histogram shows
   the number
# of words with that amount of letters, that is, I have more words with great length. Most have 6 letters.
dev.off()
setwd(working_directory)

# CODES BELOW, UNFORTUNATELY ARE NOT WORKING! (BECAUSE OF PROBLEMS WITH qdap PACKAGE)
#install.packages("stringr")
library(stringr)

# Graph with the distribution of the characters and the proportion in which they appear
#words %>%
#  str_split("") %>%
#  sapply(function(x) x[-1]) %>%
#  unlist %>%
#  dist_tab %>%
```

```
#  mutate(Letter=factor(toupper(interval),
#                       levels=toupper(interval[order(freq)]))) %>%
#  ggplot(aes(Letter, weight=percent)) +
#  geom_bar() +
#  coord_flip() +
#  labs(y="Proporção") +
#  scale_y_continuous(breaks=seq(0, 12, 2),
#                     label=function(x) paste0(x, "%"),expand=c(0,0), limits=c(0,12))

# Heatmap graph showing the position of the character and the proportion in which it appears in that position
#words %>%
#  lapply(function(x) sapply(letters, gregexpr, x, fixed=TRUE)) %>%
#  unlist %>%
#  (function(x) x[x!=-1]) %>%
#  (function(x) setNames(x, gsub("\\d", "", names(x)))) %>%
#  (function(x) apply(table(data.frame(letter=toupper(names(x)),
#                                      position=unname(x))),
#                     1, function(y) y/length(x))) %>%
#  qheat(high="green", low="yellow", by.column=NULL,
#        values=TRUE, digits=3, plot=FALSE) +
#  labs(y="Letra", x="Posição") +
#  theme(axis.text.x=element_text(angle=0)) +
#  guides(fill=guide_legend(title="Proporção"))

# Preserving the job done until here ----------------------------------------------------------------------------------------------------
# GOOD PRACTICES: Converting dtms matrix to a csv file and saving in my SO. In this way I'm preserving the job done until here.
# dtms is the matrix with less sparse terms that I'll use to perform Machine Learning
setwd(results_directory)
m <- as.matrix(dtms)
dim(m)
write.csv(m, file="dtms.csv")
setwd(working_directory)

# Cleaning Memory
rm('df', 'dfSource', 'docs', 'toSpace', 'toString', 'dtm', 'dtm_Matrix', 'freq', 'ord', 'freq_df', 'm', 'dtms', 'wf', 'dark2', 'words')

# Machine Learning ----------------------------------------------------------------------------------------------------------------------------------------
  -----

# SET WORKING DIRECTORY
#working_directory <- "C:/FCD/4MachineLearning/Cap22/R1"
#results_directory <- "C:/FCD/4MachineLearning/Cap22/R1/Results7"
setwd(working_directory)

# GETTING CURRENT DIRECTORY
getwd()

# Dataframe to store results
dfResults <- data.frame(Model = character(),
                        Accuracy = numeric(),
                        p_value = numeric(),
```

```
                Most_Important_Features = character(),
                Least_Important_Features = character())

# Datasets ----------------------------------------------------------------------------------------------------------
# Let's start getting back the matrix after all pre - processing job done
setwd(results_directory)
DMdf <- read.csv('dtms.csv', encoding = "utf8" )
DMdf$X. <- NULL
setwd(working_directory)
# Let's see the dimension of this dataset (I MUST have 138 lines)
dim(DMdf)
#View(DMdf)
# Renaming first column to match the name "ID" when I merge this and the "training_variants" dataframe
colnames(DMdf)[1]  <- "ID"

# Loading Dataset with the description of the genetic mutations
dfTrain_variants <- read.csv("training_variants")
# Verifying if there's NA in the dataset: NO, there is not NA
sum(is.na(dfTrain_variants))

# Merging Both Datasets by ID
MLdf <- merge(DMdf, dfTrain_variants, by = 'ID' )
#View(MLdf)
dim(MLdf)
str(MLdf)
str(MLdf[100:111])
# As observed above, some columns are in the wrong type so I'll adjust it

####
# I'VE NOTICED A PROBLEM DURING CLEANING PROCESS
# Looking at the amount of each Class
ggplot(data = MLdf, aes(x = Class)) +
  geom_bar() +
  geom_text(stat='count', aes(label=..count..), vjust=-1) +
  ggtitle('Amount of each Class (Some Problems)')
setwd(results_directory)
ggsave('9-Amount of each Class.png', width = 10, height = 6)
setwd(working_directory)

# We've got 3 problems here:
#  - Classes 3, 6, 8 have just one line in the dataset. This will be a problem on training or predictions because only one feature might
    be used.
#    So, I'll not consider these 3 classes from now on
MLdf <- MLdf %>%
  filter(Class != 3) %>%
  filter(Class != 6) %>%
  filter(Class != 8)

# Looking at the Results
ggplot(data = MLdf, aes(x = Class)) +
  geom_bar() +
```

```
  geom_text(stat='count', aes(label=..count..), vjust=-1) +
  ggtitle('Amount of each Class (Correct)')
setwd(results_directory)
ggsave('10-Amount of each Class.png', width = 10, height = 6)
setwd(working_directory)

# From now on we'll have 135 lines to use in the Machine Learning Model AND 6 Classes to predict
dim(MLdf)
####

# Adjusting some columns
str(MLdf[100:111])
MLdf$Gene <- as.numeric(MLdf$Gene)
MLdf$Variation <- as.numeric(MLdf$Variation)
MLdf$Class <- as.factor(MLdf$Class)
# dim(MLdf)[2] -> this command will get automatically the number of the last column where I have the "Class" and I'm changing its
   name to "target"
dim(MLdf)[2]
colnames(MLdf)[dim(MLdf)[2]]  <- "target"
# Removing "ID" column
MLdf$ID <- NULL

# Final Results
str(MLdf[100:110])

# Cleaning Memory
rm( 'DMdf', 'dfTrain_variants')

# Univariate Analysis ----------------------------------------------------------------
# Target variable frequency
freq_target <- cbind( Freq = table(MLdf$target),
                      Cumul = cumsum(table(MLdf$target)),
                      relative = round((prop.table(table(MLdf$target))*100),2))
freq_target <- as.data.frame(freq_target)
freq_target
# Based on this information I will build a Pie Chart

# Pie Chart for the target variable
# freq_target$relative -> acquiring the values of this column
slices <- c(freq_target$relative)
lbls <- c("Class 1", "Class 2", "Class 4", "Class 5", "Class 7", "Class 9")
setwd(results_directory)
png('11-Class_Distribution.png', width = 700, height = 700, res = 100)
pie(slices, labels = lbls, clockwise = TRUE, main = "Class Distribution", col = brewer.pal(9, "YlGn") )
# We have most Classes 1 and 4
dev.off()
setwd(working_directory)

# Cleaning Memory
rm( 'freq_target', 'slices', 'lbls')
```

```
# Scaling Features ----------------------------------------------------------------
# Getting all features EXCEPTING "target"
# dim(MLdf)[2] -> this command will get automatically the number of total columns
# -1 -> because I'm excluding "target" variable from the scale command
colnames(MLdf)[1:dim(MLdf)[2]-1]
# Scale on Numerical Features
MLdf_scaled <- scale(MLdf[1:dim(MLdf)[2]-1])
# Merging with "target" feature
MLdf_scaled <- cbind(MLdf[dim(MLdf)[2]], MLdf_scaled)
#View(MLdf_scaled)
dim(MLdf_scaled)

# Train_Test Split ----------------------------------------------------------------
# Acquiring 60% of the MLdf data. In this case I'm extracting the indexes to...
set.seed(666)
indexes <- sample(1:nrow(MLdf_scaled), size = 0.6 * nrow(MLdf_scaled))
# ...use 60% of them as train and to...
train.data <- MLdf_scaled[indexes,]
# ...use 40% of them as test.
test.data <- MLdf_scaled[-indexes,]
# Classes of these objects
class(train.data)
class(test.data)
# These classes are dataframes, that is, when I arrive at the end of all model creation and
# evaluation process, I will present new data for the chosen model. These new data must be in
# the same class as the train and test data !!!!!!!!!!!!!!!!!!!!!!

# Cleaning Memory
rm('indexes')

# Correlation between predictor variables ----------------------------------------------------------
# Note that the target variable has been excluded from this correlation
# names(train.data) -> returns the names of all the columns without me having to write each one. A smart way!
descrCor <- cor(train.data[,names(train.data) != "target"])
descrCor
setwd(results_directory)
png('12-Correlation.png', width = 1500, height = 1500, res = 100)
levelplot(descrCor,
      main = paste("Correlation between Features"),
      scales = list(x = list(rot = 90), cex = 1.0))
# As correlações são baixas e muito parecidas, o que não indica Multicolinearidade, podemos prosseguir!
# Observe que Variation and Gene são altamente correlacionadas com todas as outras variáveis, o que indica que cada variável
  explica variações genéticas.
dev.off()
setwd(working_directory)

# Cleaning Memory
rm('descrCor')

# Model 01 - nnet ----------------------------------------------------------------------------
#install.packages("nnet")
```

```
library(nnet)

# Creating Model
# Building a Logistic Regression Model with all features
# Building a string with target and features
formula.init <- "target ~ ."
# Fit the model
set.seed(666)
model_v1 <- nnet::multinom(formula.init, data = train.data)
# Summarize the model
summary(model_v1)
# Make predictions
predicted.classes <- predict(model_v1, test.data)
head(predicted.classes)
# Model accuracy
# Here I present the predictions (data) and compare with the original reference (reference) separated earlier. Note: positive value
  will receive "1"
# Note that the model's accuracy is in "Accuracy", the higher the better. (Ranges from 0.0 to 1.0)
confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')

# Our Accuracy with this model is: 0.7037

# Most Important Variables
formula <- "target ~ ."
formula <- as.formula(formula)
# Using trainControl package I will create the control object with the specification of a Cross-Validation method "repeatedcv"
# At the end of various repetitions, it will be possible to extract the most relevant variables for the model
control <- trainControl(method = "repeatedcv", number = 10, repeats = 2)
# Here, perform the repetitions with Cross-Validation training a glm model with ALL variables
model_var_import <- train(formula, data = train.data, method = "multinom", trControl = control)
# Finally I acquire the variables of importance
importance <- varImp(model_var_import, scale = FALSE)
# Plot of Most Important Variables
plot(importance)

# As we can see:
#  - human, mutate, Variation, negative, tumor are among the best features

# Storing this Model Results:
setwd(results_directory)

# Plot Var_Importance
png('13-Model_v1_Var_Importance.png', width = 900, height = 1200, res = 100)
plot(importance)
dev.off()

# Model Name
model <- "Neural Networks"
# Model Accuracy and P-Value
confMat <- confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')
Acc <- round(confMat[["overall"]][["Accuracy"]], 6)
```

```
p <- round(confMat[["overall"]][["AccuracyPValue"]], 7)
# Most Important Features
import_var_df <- as.data.frame(importance[["importance"]])
import_var <- import_var_df %>%
  arrange(desc(Overall)) %>%
  head(10) %>%
  rownames() %>%
  toString()
# Least Important Features
not_import_var <- import_var_df %>%
  arrange(desc(Overall)) %>%
  tail(10) %>%
  rownames() %>%
  toString()

# Combining the above Results
dfResults1 <- data.frame(model, Acc, p, import_var, not_import_var)
colnames(dfResults1) <- c("Model", "Accuracy", "p_value", "Most_Important_Features", "Least_Important_Features")
dfResults <- rbind(dfResults, dfResults1)
#colnames(dfResults) <- c("Model", "Accuracy", "p_value", "Most_Important_Features", "Least_Important_Features")

setwd(working_directory)

# Cleaning Memory
rm('formula.init', 'predicted.classes', 'formula', 'control', 'model_var_import', 'importance', 'confMat', 'model', 'Acc', 'p',
   'import_var_df',
   'import_var', 'not_import_var', 'dfResults1')

# Model 02 - Decision Tree ----------------------------------------------------------------
#install.packages("rpart")
library(rpart)

# Building a Decision Tree Model with all features
# Building a string with target and features
formula.init <- "target ~ ."
set.seed(666)
model_v2 <- rpart(formula = formula.init,
                  data = train.data,
                  method = 'class')
# Summarize the model
summary(model_v2)

# Preparing for next 2 plots (1x2, i.e. 1 line and 2 columns)
par(mfrow = c(1, 2))

# Plot of Decision Tree Branches
plot(model_v2, uniform = TRUE, main = "Classification Tree")
text(model_v2, use.n = TRUE, all = TRUE, cex = .8)
printcp(model_v2)

# Checking through the percentage of errors, if I need to do the Pruning because this model learned too much (Overfitting)
```

```
# Errors
model_v2$cptable[which.min(model_v2$cptable[,"xerror"]),"CP"]
plotcp(model_v2)
# --> Won't be necessary to Prune the Tree

# Back to Normal Plots
par(mfrow = c(1, 1))

# Make predictions
predicted.classes <- predict(model_v2, test.data, type = "class")

# Model accuracy
# Here I present the predictions (data) and compare with the original reference (reference) separated earlier. Note: positive value
  will receive "1"
# Note that the model's accuracy is in "Accuracy", the higher the better. (Ranges from 0.0 to 1.0)
confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')

# Our Accuracy with this model is: 0.4630

# and Most Important Variables:
#  - analysis, demonstrate, include, functional, method, cell in this order.

# Storing this Model Results:
setwd(results_directory)

# Plot Var_Importance
png('14-Model_v2_Classification_Tree.png', width = 2000, height = 900, res = 100)
# Preparing for next 2 plots (1x2, i.e. 1 line and 2 columns)
par(mfrow = c(1, 2))
# Plot of Decision Tree Branches
plot(model_v2, uniform = TRUE, main = "Classification Tree")
text(model_v2, use.n = TRUE, all = TRUE, cex = .8)
printcp(model_v2)
# Checking through the percentage of errors, if I need to do the Pruning because this model learned too much (Overfitting)
# Errors
model_v2$cptable[which.min(model_v2$cptable[,"xerror"]),"CP"]
plotcp(model_v2)
# --> Won't be necessary to Prune the Tree
# Back to Normal Plots
par(mfrow = c(1, 1))
dev.off()

# Model Name
model <- "Decision Tree"
# Model Accuracy and P-Value
confMat <- confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')
Acc <- round(confMat[["overall"]][["Accuracy"]], 6)
p <- round(confMat[["overall"]][["AccuracyPValue"]], 7)
# Most Important Features
import_var_df <- as.data.frame(model_v2[["variable.importance"]])
import_var <- import_var_df %>%
```

```
  head(10) %>%
  rownames() %>%
  toString()
# Least Important Features
not_import_var <- import_var_df %>%
  tail(10) %>%
  rownames() %>%
  toString()

# Combining the above Results
dfResults1 <- data.frame(model, Acc, p, import_var, not_import_var)
colnames(dfResults1) <- c("Model", "Accuracy", "p_value", "Most_Important_Features", "Least_Important_Features")
dfResults <- rbind(dfResults, dfResults1)
#colnames(dfResults) <- c("Model", "Accuracy", "p_value", "Most_Important_Features", "Least_Important_Features")

setwd(working_directory)

# Cleaning Memory
rm('formula.init', 'predicted.classes', 'confMat', 'model', 'Acc', 'p', 'import_var_df', 'import_var', 'not_import_var', 'dfResults1')

# Model 03 - KNN ----------------------------------------------------------------
# Building a KNN Model with all features
# Building a string with target and features
formula.init <- "target ~ ."

# Using trainControl package I will create the control object with the specification of a Cross-Validation method "repeatedcv"
control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

# Now building the model
# train is a function of the caret package that performs the training of several models. I specify the model I want in method = 'knn'
# Attention: I have not determined the metric to be used by the model. By default it uses 'accuracy'. Below I will create a new model
  with another metric to evaluate the performance
set.seed(3)
model_v3 <- train(target ~ .,
                  data = train.data,
                  method = "knn",
                  trControl = control,
                  # preProcess = c("center","scale"), # This is Normalization and I've already done before
                  tuneLength = 20)
# Summarize the model
summary(model_v3)

# Looking at each k value and the optimun k
model_v3

# The best k:
model_v3[["finalModel"]][["k"]]
# with maximum accuracy of:
max(model_v3[["results"]][["Accuracy"]])

# Neighbors x Accuracy
```

```
ggplot(model_v3)
#plot(model_v3, print.thres = 0.5, type="S")

# Important Features for each Class
plot(varImp(model_v3, scale = FALSE))

# Most Important Features by mean
import_var_df <- varImp(model_v3, scale = FALSE)
import_var_df <- import_var_df[["importance"]]
import_var_df$mean <- apply(import_var_df, 1, mean)

# Make predictions
predicted.classes <- predict(model_v3, test.data)
head(predicted.classes)
# Model accuracy
# Here I present the predictions (data) and compare with the original reference (reference) separated earlier. Note: positive value
  will receive "1"
# Note that the model's accuracy is in Accuracy, the higher the better. (Ranges from 0.0 to 1.0)
confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')

# Our Accuracy with this model is: 0.5926

# and Most Important Variables:
#  - activity, functional, method, demonstrate, expression, loss in this order.

# Storing this Model Results:
setwd(results_directory)

# Plots
png('15-Model_v3_K_Accuracy.png', width = 2000, height = 900, res = 100)
# Neighbors x Accuracy
ggplot(model_v3)
dev.off()

png('16-Model_v3_Var_Importance.png', width = 2000, height = 750, res = 100)
# Important Features for each Class
plot(varImp(model_v3, scale = FALSE))
dev.off()

# Model Name
model <- "K-Nearest Neighbors"
# Model Accuracy and P-Value
confMat <- confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')
Acc <- round(max(model_v3[["results"]][["Accuracy"]]), 6)
p <- round(confMat[["overall"]][["AccuracyPValue"]], 7)
# Most Important Features
import_var_df <- import_var_df
import_var <- import_var_df %>%
  arrange(desc(mean)) %>%
  head(10) %>%
  rownames() %>%
```

```
 toString()
# Least Important Features
not_import_var <- import_var_df %>%
 arrange(desc(mean)) %>%
 tail(10) %>%
 rownames() %>%
 toString()

# Combining the above Results
dfResults1 <- data.frame(model, Acc, p, import_var, not_import_var)
colnames(dfResults1) <- c("Model", "Accuracy", "p_value", "Most_Important_Features", "Least_Important_Features")
dfResults <- rbind(dfResults, dfResults1)
#colnames(dfResults) <- c("Model", "Accuracy", "p_value", "Most_Important_Features", "Least_Important_Features")

setwd(working_directory)

# Cleaning Memory
rm('formula.init', 'control', 'predicted.classes', 'confMat', 'model', 'Acc', 'p', 'import_var_df', 'import_var', 'not_import_var',
   'dfResults1')

# Model 04 - Random Forest ----------------------------------------------------------------
#install.packages("randomForest")
library(randomForest)
# In order to analyze and avoid underfitting and overfitting, I will test various ntree in the model to obtain the most suitable RMSE.

model_v4 <- function(n){
 set.seed(666)
 model <- randomForest(target ~ .,
                       data = train.data,
                       ntree = n,
                       nodesize = 10)

 predicted.classes <- predict(model_v3, test.data)
 accuracy <- confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')

 return(accuracy[["overall"]][["Accuracy"]])
}

# Building a table in order to store the accuracy values for analysis
tabACC <- data.frame(ntree = seq(5,85,5))
Results <- c()

# Control function
for (i in tabACC$ntree) {
 Results <- append(Results, model_v4(i))
}

# Joining Results and Analyzing
tabACC <- cbind(tabACC, Results)

# Graphical Analysis
```

```
colnames(tabACC) <- c('ntree', 'Accuracy')
ggplot(tabACC, aes(x = ntree, y = Accuracy)) +
  geom_point() +
  stat_smooth(method = 'lm', formula = y ~ poly(x,10), se= FALSE) +
  labs(title = "Random Forest", x = "ntree", y = 'Accuracy') + guides(color = 'none') + theme_light()

# Maximum Accuracy and ntree
maxACC <- max(tabACC$Accuracy)
# Which lines of the dataframe tabACC have the greatest Accuracy
line <- which(tabACC$Accuracy == maxACC )
line
tabACC[line, ]
# ntree
ntree = min(tabACC[line, ]$ntree)

# Now building the best Random Forest model
set.seed(666)
model_v4 <- randomForest(target ~ .,
                         data = train.data,
                         ntree = ntree,
                         nodesize = 10)
# Summarize the model
model_v4
summary(model_v4)

# Make predictions
predicted.classes <- predict(model_v4, test.data)
head(predicted.classes)
# Model accuracy
# Here I present the predictions (data) and compare with the original reference (reference) separated earlier. Note: positive value
  will receive "1"
# Note that the model's accuracy is in Accuracy, the higher the better. (Ranges from 0.0 to 1.0)
confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')

# Our Accuracy with this model is: 0.6667

# and Most Important Variables:
#  - demonstrate, wild, cell, activity, mutate, functional, mutation in this order.

# Assessing the importance of all variables
# CREATING A MODEL WITH randomForest AND AFTER EXTRACTING THE MOST RELEVANT VARIABLES, BECAUSE importance IS SET
  TO TRUE
model_var_import <- randomForest(target ~ . ,
                                 data = train.data,
                                 ntree = ntree,
                                 nodesize = 10,
                                 importance = TRUE)

# Plotando as variáveis por grau de importância
varImpPlot(model_v4, color = 'seagreen')
```

```
# Storing this Model Results:
setwd(results_directory)

# Plots
png('17-Model_v4_Accuracy_ntree.png', width = 1500, height = 900, res = 100)
ggplot(tabACC, aes(x = ntree, y = Accuracy)) +
  geom_point() +
  stat_smooth(method = 'lm', formula = y ~ poly(x,10), se= FALSE) +
  labs(title = "Random Forest", x = "ntree", y = 'Accuracy') + guides(color = 'none') + theme_light()
dev.off()

png('18-Model_v4_Var_Importance.png', width = 1000, height = 1000, res = 100)
# Important Features for each Class
varImpPlot(model_v4, color = 'seagreen')
dev.off()

# Model Name
model <- "Random Forest"
# Model Accuracy and P-Value
confMat <- confusionMatrix(table(data = predicted.classes, reference = test.data$target), positive = '1')
Acc <- round(confMat[["overall"]][["Accuracy"]], 6)
p <- round(confMat[["overall"]][["AccuracyPValue"]], 7)
# Most Important Features
import_var_df <- as.data.frame(varImpPlot(model_v4))
import_var <- import_var_df %>%
  arrange(desc(MeanDecreaseGini)) %>%
  head(10) %>%
  rownames() %>%
  toString()
# Least Important Features
not_import_var <- import_var_df %>%
  arrange(desc(MeanDecreaseGini)) %>%
  tail(10) %>%
  rownames() %>%
  toString()

# Combining the above Results
dfResults1 <- data.frame(model, Acc, p, import_var, not_import_var)
colnames(dfResults1) <- c("Model", "Accuracy", "p_value", "Most_Important_Features", "Least_Important_Features")
dfResults <- rbind(dfResults, dfResults1)
#colnames(dfResults) <- c("Model", "Accuracy", "p_value", "Most_Important_Features", "Least_Important_Features")

setwd(working_directory)

# Cleaning Memory
rm('i', 'tabACC', 'Results', 'maxACC', 'line', 'ntree', 'predicted.classes', 'model_var_import', 'confMat', 'model', 'Acc', 'p',
   'import_var_df', 'import_var', 'not_import_var', 'dfResults1')

# Model 05 - XGBoost ----------------------------------------------------------------
#install.packages("xgboost")
library(xgboost)
```

```
# XGBoost needs some exclusive configurations, let's start by making a copy of our original dataset
xgbDF <- MLdf_scaled

# Getting just "target" column and the number of classes in it. num_class will be used later by our Model to predict multiclass features
target <- xgbDF$target
num_class <- length(levels(target))

# Here is an important step, XGBoost must receive the classes to predict in an integer format starting by 0.
# So, I'm converting the target feature into integer, where the first class will be 0 and the second will be 1 and so on.
# As it starts with 0, I need to remove the last convertion (this is the reason "-1" appears below)
label <- as.integer(xgbDF$target) - 1

# Removing target feature from original dataset (xgbDF). I'll use "label" object from now on because it is in the right format, integer format
xgbDF$target <- NULL

# Train_Test Split
# Acquiring 60% of the MLdf data. In this case I'm extracting the indexes to...
set.seed(666)
indexes <- sample(1:nrow(xgbDF), size = 0.6 * nrow(xgbDF))
# ...use 60% of them as train and to...
train.data <- as.matrix(xgbDF[indexes,])
train.label <- label[indexes]
# ...use 40% of them as test.
test.data <- as.matrix(xgbDF[-indexes,])
test.label <- label[-indexes]
# Classes of these objects
class(train.data)
class(test.data)

# To use XGBoost it is necessary to use a Dense Matrix
xgb.train <- xgb.DMatrix(data = train.data, label = train.label)
xgb.test <- xgb.DMatrix(data = test.data, label = test.label)

# Setting up some parameters before building our model
# booster -> which booster to use, can be gbtree or gblinear
# -> THE NEXT PARAMETERS ARE FOR TREE BOOSTER (gbtree). IF LINEAR BOOSTER (gblinear) REQUIRED THEN GO TO DOCUMENTATION TO SEE WHICH PARAMETERS TO SET UP <-
# eta -> control the learning rate: scale the contribution of each tree by a factor of 0 < eta < 1.
#       Used to prevent overfitting by making the boosting process more conservative. Lower value for eta implies
#       larger value for nrounds: low eta value means model more robust to overfitting but slower to compute. Default: 0.3
# max_depth -> maximum depth of each Decision Tree. Default: 6
# gamma -> minimum loss reduction required to make a further partition on a leaf node of the tree. the larger, the more conservative
#        the algorithm will be.
# subsample -> subsample ratio of the training instance. Setting it to 0.5 means that xgboost randomly collected half of
#            the data instances to grow trees and this will prevent overfitting. It makes computation shorter (because less data
#            to analyse). It is advised to use this parameter with eta and increase nrounds. Default: 1
# colsample_bytree -> subsample ratio of columns when constructing each tree. Default: 1
```

```
# -> THE NEXT PARAMETERs ARE TASK PARAMETERS <-
# objective -> specify the learning task and the corresponding learning objective, users can pass a self-defined function to it.
#            this parameter has lots of configurations, looking at the documentation can bring more benefits in the choice.
# eval_metric -> evaluation metrics for validation data. Metric will be assigned according to objective but users can pass another
  metrics.
#             By default it is used "error" for classification but the section "Details" of the documentation has another metrics
# num_class -> set the number of classes. To use only with multiclass objectives (OUR CASE HERE).
params = list(booster="gbtree",
          eta = 0.001,
          max_depth = 6,
          gamma = 3,
          subsample = 0.75,
          colsample_bytree = 1,
          objective = "multi:softprob",
          eval_metric = "mlogloss",
          num_class = num_class)

# watchlist -> named list of xgb.DMatrix datasets to use for evaluating model performance. Metrics specified in either eval_metric.
#            With watchlist, I don't need to calculate the error rate for the test data because it already returns this information to me.
watchlist <- list(train = xgb.train, test = xgb.test)
watchlist

# xgb.train -> It's an advanced interface for training an xgboost model. The xgboost function is a simpler wrapper for xgb.train.
# data -> training dataset. xgb.train accepts only an xgb.DMatrix as the input.
# params -> Previously configured.
# nround -> max number of boosting iterations
# nthread -> Number of threads, this is the number of Parallelization used to execute the model
# early_stopping_rounds -> If NULL, the early stopping function is not triggered. If set to an integer k, training with a validation set
#                    will stop if the performance doesn't improve for k rounds
# verbose -> If 0, xgboost will stay silent. If 1, it will print information about performance. If 2, some additional information will be
  printed out.
# watchlist -> Previously configured.
set.seed(666)
model_v5 <- xgb.train(data = xgb.train,
                params = params,
                nround = 1000,
                nthread = 1,
                early_stopping_rounds = 10,
                verbose = 1,
                watchlist = watchlist)
# Summarize the model
model_v5
summary(model_v5)

# Make Predicitions
# reshape -> used this parameter because the predict function gives a list of numbers and the reshape = TRUE will convert it back
  to a matrix
predicted.classes <- predict(model_v5, test.data, reshape=T)
# With the reshape done above, I'm able to transform a matrix into a dataframe
predicted.classes <- as.data.frame(predicted.classes)
# Renaming my columns with the name of each class
```

```
colnames(predicted.classes) <- levels(target)

# Building one more column in the predicted.classes dataframe. Here we are using the apply function to iterate over the "predicted.classes"
# dataframe, applying the function (function(x)) by rows (number 1 in the function, if 2 it would do by column). The function (function(x)) is
# looking for the highest probability by row (max(x)) and returning the number of that column (using the function which), so with this number
# I'm extracting the name of that column (colnames(predicted.classes)[which.max(x)]) of this dataframe. The result is the number of the class
# with the highest probability. I'm storing this number in the column "prediction"
predicted.classes$prediction <- apply(predicted.classes, 1, function(x) colnames(predicted.classes)[which.max(x)])

# Building one more column in the predicted.classes dataframe.
# [test.label+1] -> test.label is the number from 0 to 5 acquired in the beggining of the XGBoost modeling. This is the sequencial number required
#                   by the model to predict multiclass problems. The "+1" is because test.label starts with 0. This will return the number of the
#                   original classes to be predicted.
# levels(target)[ ] -> with the number above I'm searching for the corresponding original number of each class, i.e.:
#                      levels(target) = "1" "2" "4" "5" "7" "9"
#                      levels(target)[1] = 1
#                      levels(target)[2] = 2
#                      levels(target)[3] = 4 .....
predicted.classes$target <- levels(target)[test.label + 1]

# Looking at the Confusion Matrix Table
table(data = predicted.classes$prediction, reference = predicted.classes$target)
# Class 5 didn't appear in the predicted.classes$prediction, so I'll remove it to use the confusionMatrix() function from caret
predicted.classes <- filter(predicted.classes, target != 5)
confusionMatrix(table(data = predicted.classes$prediction, reference = predicted.classes$target))

# Calculate and printing the final results by comparing the success between prediction and target.
finalResult <- sum(predicted.classes$prediction == predicted.classes$target) / nrow(predicted.classes)
print(paste("Accuracy XGBoost =", sprintf("%1.2f%%", 100*finalResult)))

# Our Accuracy with this model is: 0.7115

# and Most Important Variables:
#  - functional, mutate, human, activity, require, express, suggest mutation in this order.

# Feature Importance
import_var_df <- xgb.importance(model = model_v5)
import_var_df

# Storing this Model Results:
setwd(results_directory)

# Plots
#install.packages("DiagrammeR")
library(DiagrammeR)
```

```
png('19-Model_v5_Deepness.png', width = 2000, height = 900, res = 100)
# Plotting Deepness
xgb.plot.deepness(model_v5)
dev.off()

# Plotting First Trees
#xgb.plot.tree(model = model_v5, trees = 0:4)
#install.packages("DiagrammeRsvg")
#install.packages("rsvg")
library(DiagrammeRsvg)
library(rsvg)
xgbtree <- xgb.plot.tree(model = model_v5, trees = 0:4, render = FALSE)
export_graph(xgbtree, '20-Model_v5_XGBoost_Trees.png', width = 2000, height = 1500)

# Model Name
model <- "XGBoost"
# Model Accuracy and P-Value
confMat <- confusionMatrix(table(data = predicted.classes$prediction, reference = predicted.classes$target))
Acc <- round(confMat[["overall"]][["Accuracy"]], 6)
p <- round(confMat[["overall"]][["AccuracyPValue"]], 7)
# Most Important Features
import_var_df <- xgb.importance(model = model_v5)

#install.packages("tidyverse")
library(tidyverse)  # to use column_to_rownames() function to move a feature to rownames so it's possible to do the '%>%
  rownames()' below
import_var_df <- import_var_df %>%
            column_to_rownames(., var = "Feature")

import_var <- import_var_df %>%
 head(10) %>%
 rownames() %>%
 toString()
# Least Important Features
not_import_var <- import_var_df %>%
 tail(10) %>%
 rownames() %>%
 toString()

# Combining the above Results
dfResults1 <- data.frame(model, Acc, p, import_var, not_import_var)
colnames(dfResults1) <- c("Model", "Accuracy", "p_value", "Most_Important_Features", "Least_Important_Features")
dfResults <- rbind(dfResults, dfResults1)
#colnames(dfResults) <- c("Model", "Accuracy", "p_value", "Most_Important_Features", "Least_Important_Features")

setwd(working_directory)

# Cleaning Memory
rm('xgbDF', 'target', 'num_class', 'label', 'indexes', 'train.data', 'train.label', 'test.data', 'test.label', 'xgb.train', 'xgb.test',
  'params', 'watchlist', 'predicted.classes', 'finalResult', 'xgbtree', 'model', 'confMat', 'Acc', 'p', 'import_var_df', 'import_var',
  'not_import_var',
```

```
  'dfResults1')

# Final Thoughts and Adjustments ----------------------------------------------------------------------------------------------------
# Storing Models Results in csv format:
setwd(results_directory)
write.csv(dfResults, file = 'Final_Results.csv')

# ATTENTION:
# The error below happens in the geom_bar () ggplot plots, explanation:
#"Error: stat_count() must not be used with a y aesthetic."
#This error comes due to the mapping of elements in aes() while plotting barplot.
# geom_bar by default takes a count of values as y in a bar plot. So use stat="identity" within barplot() to avoid the default and use
  fields as used in mapping function.
# ggplot(data,aes(field1,field2)) + geom_plot(stat="identity")

png('21-Final_Results.png', width = 900, height = 500, res = 100)
# Looking at Accuracy by Model
ggplot() +
 geom_bar(data = dfResults,
       aes(x = Model, y = Accuracy),
       stat = 'identity') +
 coord_flip() +
 ggtitle("Models x Accuracy")

dev.off()

# Analysis of Most Important Features according to Models ---------------------------------------------------------------------------
# Preparing a DataFrame to use in the Corpus ---

# Getting "Most_Important" column from dfResults and spliting this string by ", "
spliting_features <- str_split(dfResults$Most_Important_Features, ", ")
# Converting to dataframe makes each word splitted above stay in a line
spliting_features1 <- as.data.frame(spliting_features[[1]])
colnames(spliting_features1) <- c('Text')
spliting_features2 <- as.data.frame(spliting_features[[2]])
colnames(spliting_features2) <- c('Text')
spliting_features3 <- as.data.frame(spliting_features[[3]])
colnames(spliting_features3) <- c('Text')
spliting_features4 <- as.data.frame(spliting_features[[4]])
colnames(spliting_features4) <- c('Text')
spliting_features5 <- as.data.frame(spliting_features[[5]])
colnames(spliting_features5) <- c('Text')
# Binding all rows
most_Important_df <- rbind(spliting_features1, spliting_features2, spliting_features3, spliting_features4, spliting_features5)
# Adding column "ID" to "most_Important_df"< that"s a requirement do prepare a Corpus
most_Important_df$ID <- 1:nrow(most_Important_df)
#View(most_Important_df)
# Simplyfing the name of the dataframe
df <- most_Important_df
# We must have 50 lines
nrow(df)
```

```
# Preparing a Corpus ---
# To use the function DataframeSource I need to transform my dataframe into some columns (doc_id, text, and metadatas)
# stringsAsFactors -> IMPORTANT parameter so my dataframe(df) stay as configured above
dfSource <- data.frame(doc_id = df$ID,
                       text = df$Text,
                       stringsAsFactors = FALSE)
str(dfSource)
dfSource$text <- as.character(dfSource$text)
# Creating a data frame source, an object required by Corpus function
dfSource <- DataframeSource(dfSource)
# Creating a Corpus object
docs <- Corpus(dfSource)
#inspect(docs)
#meta(docs)

# Quantitative Analysis ---
# Converting Corpus 'docs' to perform a Quantitative Analysis
# As Machines don't understand strings (texts), we'll need to convert them to mathematical formulas, to do so let's build a
  DocumentTermMatrix that is a Matrix with
# frequency count of each term of the documents

# DocumentTermMatrix -> A Term Document Matrix is a matrix with Documents in the lines, Terms in the columns and a frequency
  count in the cells of the Matrix
dtm <- DocumentTermMatrix(docs)

# A DocumentTermMatrix is really too much sparse (i.e. mostly empty) and because of this it is stored in a better and compacted
  representation in the system
# Looking at the amount of lines(documents) and columns(terms)
dim(dtm)
# We have 50 lines, that's correct according to the original dataset (most_Important_df)
class(dtm)
# Inspecting some rows and columns of this matrix:
inspect(dtm[1:50, 1:29])
# Look that "protein" shows up 220 time in the document 110 and that Sparsity = 67% to this amount of data
# Inspecting all Matrix
inspect(dtm)

# Let's look at the frequency of the terms in these docs
# First I need to convert this DocumentTermMatrix to an Matrix
dtm_Matrix <- as.matrix(dtm)
# Second we'll count the sum of each column
freq <- colSums(dtm_Matrix)
length(freq)
# Ordering these frequency
# order -> ordering the frequency of the terms in ascending order. Note that this function gets the number of the columns, and not
  the name, so I'll need to perform another
#       code to retrieve the names and number of terms (i.e. "freq[head(ord)]" and "freq[tail(ord)]")
ord <- order(freq)
# Lowest terms
freq[head(ord)]
```

```
# Highest terms
freq[tail(ord,100)]

# Preparing to plot "Models - Most Important Words"
# Building a Dataframe with 15 Most Important Terms
freq_df <- as.data.frame(freq[tail(ord, 15)])
# Renaming dataframe columns
colnames(freq_df) <- c('freq')
# Adding feature names of each frequency
freq_df$ feat <- rownames(freq_df)

ggplot(data = freq_df, aes(x = feat, y = freq)) +
  geom_bar(stat = "identity", fill='springgreen1') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("Models - Most Important Words")
setwd(results_directory)
ggsave("22-Models - Most Important Words.png", width = 10, height = 5)
setwd(working_directory)

# The barploting below was moved to a before analysis (to line 1396), here I'm just showing how to to the same in another way
# Barploting the frequency of the words
# Acquiring the Frequency of Words listing in descending order
freq <- sort(colSums(as.matrix(dtm)), decreasing = TRUE)
head(freq, 15)
# Acquiring the name of each word and its frequency, placing it in a data.frame to be able to plot with ggplotot
wf <- data.frame(word=names(freq), freq=freq)
head(wf)
# Ploting with ggplot2 frequencies above 2000
subset(wf, freq>0) %>%
  ggplot(aes(word, freq)) +
  geom_bar(stat = "identity", fill='yellowgreen') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Wordcloud Visualization
#install.packages("wordcloud")
library(wordcloud)

# Getting words which MINIMUM frequency is 300
set.seed(666)
wordcloud(names(freq), freq, min.freq = 300, colors = brewer.pal(6, "RdYlGn"))

# Getting words which MINIMUM frequency is 300 (BUT SCALING THE WORDCLOUD TO A SMALLER SHAPE)
set.seed(666)
wordcloud(names(freq), freq, min.freq = 300, scale=c(5, .1), colors = brewer.pal(6, "RdYlGn"))

# Getting words which MINIMUM frequency is 300 (BUT FORCING 50% OF PROPORTION WORDS WITH 90 DEGREE ROTATION)
setwd(results_directory)
png('23-WordCloud(Most Important Words).png', width = 700, height = 700, res = 100)
set.seed(999)
c <- brewer.pal(6, "RdYlGn")
wordcloud(names(freq), freq, min.freq = 300, rot.per = 0.5, colors = c)
```

```
dev.off()
setwd(working_directory)

# Transforming dtm into a matrix, getting the names of the columns and searching for words with less than 10 characters
words <- dtm %>%
  as.matrix %>%
  colnames %>%
  (function(x) x[nchar(x) < 10])

# How many words are left?
length(words)
# Looking at 15 words
head(words, 15)
# Number of letters and how many words with that amount of letters
table(nchar(words))

# NOT WORKING!!! (because of some dependencies)
# dist_tab -> it shows a very nice table indicating the number of letters, the frequency and the percentage of the letters in the
   whole.
#dist_tab(nchar(words))

# Inserting "words" in a dataframe and plotting (basicaly I'm plotting freq column from dist_tab above)
setwd(results_directory)
png('24-Letters_x_Words (Most Important Words).png', width = 1500, height = 600, res = 100)
data.frame(nletters=nchar(words)) %>%
  ggplot(aes(x=nletters)) +
  geom_histogram(binwidth=1) +
  geom_vline(xintercept=mean(nchar(words)),
             colour="green", size=1, alpha=.5) +
  labs(x="Number of Letters", y="Number of Words")
# What we have on the graph is on the x axis the number of letters and on the y axis the number of words, the histogram shows
   the number
# of words with that amount of letters, that is, I have more words with great length. Most have 6 letters.
dev.off()
setwd(working_directory)

# CODES BELOW, UNFORTUNATELY ARE NOT WORKING! (BECAUSE OF PROBLEMS WITH qdap PACKAGE)
#install.packages("stringr")
library(stringr)

# Graph with the distribution of the characters and the proportion in which they appear
#words %>%
#  str_split("") %>%
#  sapply(function(x) x[-1]) %>%
#  unlist %>%
#  dist_tab %>%
#  mutate(Letter=factor(toupper(interval),
#                       levels=toupper(interval[order(freq)]))) %>%
#  ggplot(aes(Letter, weight=percent)) +
#  geom_bar() +
#  coord_flip() +
```

```
#  labs(y="Proporção") +
#  scale_y_continuous(breaks=seq(0, 12, 2),
#                     label=function(x) paste0(x, "%"),expand=c(0,0), limits=c(0,12))

# Heatmap graph showing the position of the character and the proportion in which it appears in that position
#words %>%
#  lapply(function(x) sapply(letters, gregexpr, x, fixed=TRUE)) %>%
#  unlist %>%
#  (function(x) x[x!=-1]) %>%
#  (function(x) setNames(x, gsub("\\d", "", names(x)))) %>%
#  (function(x) apply(table(data.frame(letter=toupper(names(x)),
#                                      position=unname(x))),
#                     1, function(y) y/length(x))) %>%
#  qheat(high="green", low="yellow", by.column=NULL,
#        values=TRUE, digits=3, plot=FALSE) +
#  labs(y="Letra", x="Posição") +
#  theme(axis.text.x=element_text(angle=0)) +
#  guides(fill=guide_legend(title="Proporção"))

# Cleaning Memory
rm('spliting_features', 'spliting_features1', 'spliting_features2', 'spliting_features3', 'spliting_features4', 'spliting_features5',
   'most_Important_df', 'df', 'dfSource', 'docs', 'dtm', 'dtm_Matrix', 'freq', 'ord', 'freq_df', 'wf', 'c', 'words')


# Analysis of Least Important Features according to Models ----------------------------------------------------------------------------
# Preparing a DataFrame to use in the Corpus ---

# Getting "Least_Important" column from dfResults and spliting this string by ", "
spliting_features <- str_split(dfResults$Least_Important_Features, ", ")
# Converting to dataframe makes each word splitted above stay in a line
spliting_features1 <- as.data.frame(spliting_features[[1]])
colnames(spliting_features1) <- c('Text')
spliting_features2 <- as.data.frame(spliting_features[[2]])
colnames(spliting_features2) <- c('Text')
spliting_features3 <- as.data.frame(spliting_features[[3]])
colnames(spliting_features3) <- c('Text')
spliting_features4 <- as.data.frame(spliting_features[[4]])
colnames(spliting_features4) <- c('Text')
spliting_features5 <- as.data.frame(spliting_features[[5]])
colnames(spliting_features5) <- c('Text')
# Binding all rows
most_Important_df <- rbind(spliting_features1, spliting_features2, spliting_features3, spliting_features4, spliting_features5)
# Adding column "ID" to "most_Important_df"< that"s a requirement do prepare a Corpus
most_Important_df$ID <- 1:nrow(most_Important_df)
#View(most_Important_df)
# Simplyfing the name of the dataframe
df <- most_Important_df
# We must have 50 lines
nrow(df)

# Preparing a Corpus ---
```

```
# To use the function DataframeSource I need to transform my dataframe into some columns (doc_id, text, and metadatas)
# stringsAsFactors -> IMPORTANT parameter so my dataframe(df) stay as configured above
dfSource <- data.frame(doc_id = df$ID,
                       text = df$Text,
                       stringsAsFactors = FALSE)
str(dfSource)
dfSource$text <- as.character(dfSource$text)
# Creating a data frame source, an object required by Corpus function
dfSource <- DataframeSource(dfSource)
# Creating a Corpus object
docs <- Corpus(dfSource)
#inspect(docs)
#meta(docs)

# Quantitative Analysis ---
# Converting Corpus 'docs' to perform a Quantitative Analysis
# As Machines don't understand strings (texts), we'll need to convert them to mathematical formulas, to do so let's build a
  DocumentTermMatrix that is a Matrix with
# frequency count of each term of the documents

# DocumentTermMatrix -> A Term Document Matrix is a matrix with Documents in the lines, Terms in the columns and a frequency
  count in the cells of the Matrix
dtm <- DocumentTermMatrix(docs)

# A DocumentTermMatrix is really too much sparse (i.e. mostly empty) and because of this it is stored in a better and compacted
  representation in the system
# Looking at the amount of lines(documents) and columns(terms)
dim(dtm)
# We have 50 lines, that's correct according to the original dataset (most_Important_df)
class(dtm)
# Inspecting some rows and columns of this matrix:
inspect(dtm[1:50, 1:29])
# Look that "protein" shows up 220 time in the document 110 and that Sparsity = 67% to this amount of data
# Inspecting all Matrix
inspect(dtm)

# Let's look at the frequency of the terms in these docs
# First I need to convert this DocumentTermMatrix to an Matrix
dtm_Matrix <- as.matrix(dtm)
# Second we'll count the sum of each column
freq <- colSums(dtm_Matrix)
length(freq)
# Ordering these frequency
# order -> ordering the frequency of the terms in ascending order. Note that this function gets the number of the columns, and not
  the name, so I'll need to perform another
#        code to retrieve the names and number of terms (i.e. "freq[head(ord)]" and "freq[tail(ord)]")
ord <- order(freq)
# Lowest terms
freq[head(ord)]
# Highest terms
freq[tail(ord,100)]
```

```
# Preparing to plot "Models - Least Important Words"
# Building a Dataframe with 15 Least Important Terms
freq_df <- as.data.frame(freq[tail(ord, 15)])
# Renaming dataframe columns
colnames(freq_df) <- c('freq')
# Adding feature names of each frequency
freq_df$ feat <- rownames(freq_df)

ggplot(data = freq_df, aes(x = feat, y = freq)) +
  geom_bar(stat = "identity", fill='lightcoral') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("Models - Least Important Words")
setwd(results_directory)
ggsave("25-Models - Least Important Words.png", width = 10, height = 5)
setwd(working_directory)

# The barploting below was moved to a before analysis (to line 1592), here I'm just showing how to to the same in another way
# Barploting the frequency of the words
# Acquiring the Frequency of Words listing in descending order
freq <- sort(colSums(as.matrix(dtm)), decreasing = TRUE)
head(freq, 15)
# Acquiring the name of each word and its frequency, placing it in a data.frame to be able to plot with ggplotot
wf <- data.frame(word=names(freq), freq=freq)
head(wf)
# Ploting with ggplot2 frequencies above 2000
subset(wf, freq>0) %>%
  ggplot(aes(word, freq)) +
  geom_bar(stat = "identity", fill='yellowgreen') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Wordcloud Visualization
#install.packages("wordcloud")
library(wordcloud)

# Getting words which MINIMUM frequency is 300
set.seed(666)
wordcloud(names(freq), freq, min.freq = 300, colors = brewer.pal(6, "RdYlGn"))

# Getting words which MINIMUM frequency is 300 (BUT SCALING THE WORDCLOUD TO A SMALLER SHAPE)
set.seed(666)
wordcloud(names(freq), freq, min.freq = 300, scale=c(5, .1), colors = brewer.pal(6, "RdYlGn"))

# Getting words which MINIMUM frequency is 300 (BUT FORCING 50% OF PROPORTION WORDS WITH 90 DEGREE ROTATION)
setwd(results_directory)
png('26-WordCloud(Least Important Words).png', width = 700, height = 700, res = 100)
set.seed(666)
c <- brewer.pal(6, "RdYlGn")
wordcloud(names(freq), freq, min.freq = 300, rot.per = 0.5, colors = c)
dev.off()
setwd(working_directory)
```

```
# Transforming dtm into a matrix, getting the names of the columns and searching for words with less than 10 characters
words <- dtm %>%
  as.matrix %>%
  colnames %>%
  (function(x) x[nchar(x) < 10])

# How many words are left?
length(words)
# Looking at 15 words
head(words, 15)
# Number of letters and how many words with that amount of letters
table(nchar(words))

# NOT WORKING!!! (because of some dependencies)
# dist_tab -> it shows a very nice table indicating the number of letters, the frequency and the percentage of the letters in the
   whole.
#dist_tab(nchar(words))

# Inserting "words" in a dataframe and plotting (basicaly I'm plotting freq column from dist_tab above)
setwd(results_directory)
png('27-Letters_x_Words (Least Important Words).png', width = 1500, height = 600, res = 100)
data.frame(nletters=nchar(words)) %>%
  ggplot(aes(x=nletters)) +
  geom_histogram(binwidth=1) +
  geom_vline(xintercept=mean(nchar(words)),
             colour="green", size=1, alpha=.5) +
  labs(x="Number of Letters", y="Number of Words")
# What we have on the graph is on the x axis the number of letters and on the y axis the number of words, the histogram shows
   the number
# of words with that amount of letters, that is, I have more words with great length. Most have 6 letters.
dev.off()
setwd(working_directory)

# CODES BELOW, UNFORTUNATELY ARE NOT WORKING! (BECAUSE OF PROBLEMS WITH qdap PACKAGE)
#install.packages("stringr")
library(stringr)

# Graph with the distribution of the characters and the proportion in which they appear
#words %>%
#  str_split("") %>%
#  sapply(function(x) x[-1]) %>%
#  unlist %>%
#  dist_tab %>%
#  mutate(Letter=factor(toupper(interval),
#                       levels=toupper(interval[order(freq)]))) %>%
#  ggplot(aes(Letter, weight=percent)) +
#  geom_bar() +
#  coord_flip() +
#  labs(y="Proporção") +
#  scale_y_continuous(breaks=seq(0, 12, 2),
```

```
#                 label=function(x) paste0(x, "%"),expand=c(0,0), limits=c(0,12))

# Heatmap graph showing the position of the character and the proportion in which it appears in that position
#words %>%
#  lapply(function(x) sapply(letters, gregexpr, x, fixed=TRUE)) %>%
#  unlist %>%
#  (function(x) x[x!=-1]) %>%
#  (function(x) setNames(x, gsub("\\d", "", names(x)))) %>%
#  (function(x) apply(table(data.frame(letter=toupper(names(x)),
#                                      position=unname(x))),
#                 1, function(y) y/length(x))) %>%
#  qheat(high="green", low="yellow", by.column=NULL,
#        values=TRUE, digits=3, plot=FALSE) +
#  labs(y="Letra", x="Posição") +
#  theme(axis.text.x=element_text(angle=0)) +
#  guides(fill=guide_legend(title="Proporção"))

# Cleaning Memory
rm('spliting_features', 'spliting_features1', 'spliting_features2', 'spliting_features3', 'spliting_features4', 'spliting_features5',
   'most_Important_df', 'df', 'dfSource', 'docs', 'dtm', 'dtm_Matrix', 'freq', 'ord', 'freq_df', 'wf', 'c', 'words')

# Leaving only what is necessary ----------------------------------------------------------------------------------------------------
time.end  <- Sys.time()

# Time Difference from beggining to end
duration <- round(time.end - time_init, 2)

# End of Processing! Go to the Results directory and analyze all the results detailed in Graphs and Tables
setwd(results_directory)
paste("End of Processing! This application took", duration, "minutes to run. Go to ", getwd()," directory and analyze all the results
   detailed in Graphs and Tables")

rm('MLdf', 'MLdf_scaled', 'results_directory', 'working_directory')
```