



UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

ESCOLA DE INFORMÁTICA APLICADA

Aplicação de Aprendizado de Máquina para Predição de Prioridade em Gestão de
Incidentes

Lucas Alves Moreira de Souza

Orientadora

Prof^ª. Kate Cerqueira Revoredo, D.Sc

RIO DE JANEIRO, RJ – BRASIL

DEZEMBRO DE 2017

Catálogo informatizada pelo autor

S719 Souza, Lucas Alves Moreira de Aplicação de
Aprendizado de Máquina para Predição de Prioridade
em Gestão de Incidentes / Lucas Alves Moreira de
Souza. -- Rio de Janeiro, 2017.
61 f.

Orientadora: Kate Cerqueira Revoredo.

Trabalho de Conclusão de Curso (Graduação) -
Universidade Federal do Estado do Rio de Janeiro,
Graduação em Sistemas de Informação, 2017.

1. Aprendizado de Máquina. 2. Análise Preditiva.
3. scikit-learn. 4. Gestão de Incidentes. 5.
Processos de Negócio. I. Revoredo, Kate Cerqueira,
orient. II. Título.

Aplicação de Aprendizado de Máquina para Predição de Prioridade em Gestão de Incidentes

Lucas Alves Moreira de Souza

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado por:

KATE CERQUEIRA REVOREDO, D.Sc (UNIRIO)

FERNANDA ARAUJO BAIÃO AMORIM, D.Sc (UNIRIO)

RIO DE JANEIRO, RJ – BRASIL.

DEZEMBRO DE 2017

Agradecimentos

Agradeço à minha mãe, Marcia, pelo apoio e pelo amor incondicional, em qualquer circunstância, e ao meu falecido pai, Wilson, que venceu uma quantidade inacreditável de dificuldades, em vida, e conseguiu sustentar e educar, da melhor forma possível, seus três filhos.

Agradeço aos meus irmãos, João Pedro e Marjorie, pelo companheirismo e pelas experiências. Nós três estamos ficando mais velhos, mas que o tempo nunca seja suficiente para nos separar; sempre podemos contar um com o outro, e sou grato por isso.

Agradeço ao corpo docente da UNIRIO, em especial, à diretora da Escola de Informática Aplicada, Prof^a Morganna, pela excelência em gestão, resolução de conflitos e disponibilidade para escutar seus alunos; também, à Prof^a Kate, minha orientadora, por todos os direcionamentos e esclarecimentos necessários para o desenvolvimento desse trabalho.

RESUMO

Muitas empresas encontram, através da Tecnologia da Informação, insumos para apoio a decisão e otimização do trabalho, como relatórios de vendas, monitoramento de níveis de serviço, entre outros. Nos últimos anos, o aprendizado de máquina tornou-se mais uma ferramenta de apoio aos níveis gerencial e executivo, através da análise preditiva. Para o negócio, é muito valioso ter controle sobre a probabilidade de um cliente adquirir um produto específico, após receber um *email marketing*, por exemplo, ou sobre as possíveis saídas de um processo de negócio, baseado no log histórico desse processo e suas diversas características.

Nesse trabalho, será apresentado o último exemplo aplicado a um cenário real de uma empresa da Espanha, onde existem problemas de priorização de incidentes, dentro de uma área de suporte ao usuário. No cenário, pode ocorrer que um incidente específico sofra alteração de prioridade, enquanto o processo já está em execução, o que dificulta o planejamento e provoca uma queda no nível de serviço, tanto por qualidade quanto por produtividade.

Como proposta de solução para esse problema, serão feitas implementações do aprendizado de máquina, utilizando algoritmos de classificação, na linguagem Python, então a precisão desses modelos será verificada e os resultados serão apresentados. Uma vez que a empresa tenha posse da análise preditiva sobre “o próximo passo” do processo, será capaz de otimizar a alocação de recursos e reduzir o impacto dessa alteração, ainda que em tempo de execução do processo.

Palavras-chave: Aprendizado de Máquina, Análise Preditiva, scikit-learn, Gestão de Incidentes, Processos de Negócio.

ABSTRACT

Many companies, through Information Technology, acquire inputs for decision support and work optimization, such as sales reports, services levels monitoring, among others. In last years, machine learning has become a powerful tool for management and executive levels, through predictive analytics. For a business, it is valuable to know about the probability of a customer acquiring a specific product after receiving an email marketing, for example, or about the possible outputs of a business process, based on the process history log and its features.

In this work, we present the last example applied to a real scenario of a company in Spain, where exists problems of incidents prioritization, within an user support area. In this scenario, a specific incident may change priority, while process is already running, which makes planning difficult and causes a service level decrease, in quality and productivity.

As a solution to this problem, machine learning implementations will be proposed, using classification algorithms in Python language; so the accuracy of these models will be verified, along with their results. Once the company has predictive analytics on the process “next step”, it is going to be able to optimize resource allocation and reduce change impact, albeit at process execution time.

Keywords: Machine Learning, Predictive Analysis, scikit-learn, Incident Management, Business Processes.

Índice

1	Introdução.....	12
1.1	Motivação.....	12
1.2	Objetivos.....	13
1.3	Organização do texto.....	14
2	Cenário de Negócio e Fundamentação Teórica.....	15
2.1	Cenário da Empresa.....	15
2.2	Fundamentos de Machine Learning.....	16
2.2.1	Métodos de Aprendizagem para Classificação.....	17
2.2.1.1	Regressão Logística.....	18
2.2.1.2	Árvore de Decisão.....	19
2.2.1.3	Random Forest.....	22
2.2.2	Variáveis Dependentes e Independentes.....	23
2.2.3	Aplicação para o Cenário de Negócio.....	24
3	Aplicação de Machine Learning: Classificação.....	25
3.1	Tecnologias do Ambiente Python.....	25
3.1.1	Plataforma Anaconda e Spyder IDE.....	25
3.1.2	Bibliotecas.....	27
3.1.2.1	scikit-learn.....	28
3.1.2.2	pandas.....	28
3.1.2.3	numpy.....	29
3.2	Execução das Máquinas.....	29
3.2.1	Base de Dados de Incidentes.....	29
3.2.2	Etapas da Execução Python.....	31
3.2.2.1	Pré-processamento de dados.....	31
3.2.2.1.1	Tratamento de Dados Ausentes.....	32
3.2.2.1.2	Codificação de Variáveis Categóricas.....	35

3.2.2.2	Feature Matrix e Vetor Dependente	37
3.2.2.3	Train-Test Split	38
3.2.2.4	Feature Scaling	38
3.2.2.5	Execução dos Classificadores	40
4	Resultados dos Algoritmos de Análise Preditiva	43
4.1	Apresentação dos Resultados	43
4.1.1	Confusion Matrix.....	43
4.1.2	Árvore de Decisão	46
4.2	Métodos de Validação e Medição de Performance	47
4.2.1	Accuracy Paradox.....	47
4.2.2	Validação Cruzada (Método k-fold).....	48
4.2.3	Importância de Atributos	53
5	Conclusão	58
5.1	Considerações Finais	58
5.2	Limitações do Projeto	59
5.3	Trabalhos Futuros	59

Índice de Tabela

Tabela 1 - Vantagens e Desvantagens de Modelos de Classificação	17
Tabela 2 - Dicionário de Dados da Base de Incidentes	30
Tabela 3 - Comparação de Resultados entre Classificações	46
Tabela 4 - Confusion Matrix da Classificação por Árvore de Decisão	47
Tabela 5 - Confusion Matrix da Simulação sem Alteração de Prioridade	47
Tabela 6 - Resultados da Validação Cruzada	53
Tabela 7 - Coeficientes Mais Importantes para Regressão Logística; Classe P1	54
Tabela 8 - Coeficientes Mais Importantes para Regressão Logística; Classe P2	55
Tabela 9 - Coeficientes Mais Importantes para Regressão Logística; Classe P3	55
Tabela 10 - Lista Ordenada de Importância de Features para Árvore de Decisão	56
Tabela 11 - Lista Ordenada de Importância de Features para Random Forest	57

Índice de Figuras

Figura 1 - Modelo de Processo de Gestão de Incidentes.....	15
Figura 2 - Equação da Função Logística.....	18
Figura 3 - Gráfico da Função Logística.....	19
Figura 4 - Processo de splitting de uma Árvore de Decisão.....	20
Figura 5 - Exemplo de uma Árvore de Decisão.....	21
Figura 6 - Exemplo de uma Árvore de Decisão para o dataset iris.....	22
Figura 7 - Funcionamento de uma Random Forest.....	23
Figura 8 - Anaconda Navigator.....	26
Figura 9 - Spyder IDE.....	27
Figura 10 - Instalação de Pacote por Prompt de Comando.....	28
Figura 11 - Código para Importação da Base CSV.....	32
Figura 12 - Resumo da Base de Dados.....	33
Figura 13 - Contagem de Valores Nulos da Base de Dados.....	33
Figura 14 - Contagem de Valores Nulos após Remoção de Colunas.....	34
Figura 15 - Resumo da Base de Dados após Tratamento de Valores Nulos.....	35
Figura 16 - Implementação para Codificação Numérica de Variáveis Categóricas.....	36
Figura 17 - Base de Dados após Codificação Numérica.....	36
Figura 18 - Exemplo de Codificação one-hot.....	37
Figura 19 - Código para Divisão de Dados de Teste e Treinamento.....	38
Figura 20 - Fórmula da Distância Euclidiana.....	39
Figura 21 - Dados de Treinamento após Feature Scaling.....	39
Figura 22 - Código para Classificação por Regressão Logística.....	40
Figura 23 - Código para Classificação por Árvore de Decisão.....	41
Figura 24 - Código para Classificação por Random Forest.....	41
Figura 25 - Confusion Matrix da Classificação por Regressão Logística.....	44
Figura 26 - Confusion Matrix da Classificação por Árvore de Decisão.....	45
Figura 27 - Confusion Matrix da Classificação Random Forest.....	45
Figura 28 - Árvore de Decisão resultante da Classificação por Árvore de Decisão.....	46
Figura 29 - Validação Cruzada 10-fold.....	49
Figura 30 - Conflito Tendência x Variância.....	50
Figura 31 - Execução da Validação Cruzada para Regressão Logística.....	50
Figura 32 - Vetor de Precisões da Validação Cruzada para Regressão Logística.....	51

Figura 33 - Execução da Validação Cruzada para Árvore de Decisão.....	51
Figura 34 - Vetor de Precisões da Validação Cruzada para Árvore de Decisão	52
Figura 35 - Execução da Validação Cruzada para Random Forest	52
Figura 36 - Vetor de Precisões da Validação Cruzada para Random Forest.....	53
Figura 37 - Coeficientes de Variáveis Independentes na Classificação por Regressão Logística	54
Figura 38 - Importância de Features para Árvore de Decisão	56
Figura 39 - Importância de Features para Random Forest	57

1 Introdução

1.1 Motivação

Nos últimos anos, a quantidade de dados produzidos e disponíveis aumentou exponencialmente. Estima-se que, em 2020, a quantidade de dados de todo o mundo alcance 44 zettabytes [1]. As organizações têm sido capazes de explorar esses dados, descobrindo informações estratégicas para a direção do negócio e, mais recentemente, capazes até de realizar previsões. Um exemplo comum é a análise preditiva aplicada a ações na bolsa de valores, para que se identifique tendências e comportamentos de mercado [2].

Para realizar essas previsões, é muito comum a utilização de ferramentas para aprendizado de máquina: em inglês, *machine learning*. Essa subárea da computação, parte do campo de Inteligência Artificial, estuda a implementação de software capaz de aprender e identificar padrões, de forma autônoma [3]. Valendo-se de métodos estatísticos e matemáticos, aliados à programação de software, o aprendizado de máquina é utilizado para resolver problemas de classificação, regressão e *clustering*, por exemplo.

As capacidades dessa tecnologia são diversas, e os maiores ganhos de sua utilização são percebidos na identificação de imagens: o reconhecimento de pessoas por foto, do Facebook, algoritmos de realidade virtual e o sensor Kinect [4], para identificação de movimentos, sem a necessidade qualquer acessório, do videogame Microsoft Xbox são alguns exemplos. Nesse último, são utilizados algoritmos baseados em *Random Forest*, um método que será abordado nesse trabalho.

Podemos afirmar que os modelos de *machine learning* são fortemente empregados para técnicas de previsão, onde classificamos em duas grandes áreas: a previsão de valores numéricos (regressão), como visto em [2], e a previsão de categorias (classificação), como visto em [4], ou seja, variáveis dependentes categóricas. Para ambas as áreas, podemos explorar uma tendência atual no campo da inteligência artificial, que é a previsão aplicada a processos de negócio.

A tecnologia sempre deu apoio aos processos, através de sistemas integrados (ERPs), automatização de atividades e, em destaque, as ferramentas de *business*

intelligence (BI); entre outras formas de apoio a processos. O BI – valendo-se de relatórios, dashboards, análises sobre bases financeiras, contábeis, clientes, recursos humanos, entre outras – foi uma grande revolução para que as empresas pudessem enxergar o real valor dos dados que ali existiam, possibilitando a tomada de decisão orientada à realidade do negócio, por muitas vezes, chamada gestão por indicadores.

Com o fenômeno “*big data*”, aliado ao aumento de poder computacional, a realidade no uso da tecnologia de apoio a decisão tornou-se diferente. Notou-se que não mais era necessário restringir o poder analítico aos relatórios de BI, mas sim que essas novas características da computação possibilitavam o trabalho de predição.

Em posse dessa nova capacidade, a indústria e o mercado têm aplicado a predição para descoberta em processos de negócio; em foco desse trabalho, mais especificamente, a predição de prioridade de um incidente, uma vez que podemos utilizar dados históricos como unidade de negócio, centro responsável, autor do incidente, entre outros. Logo, seria possível, considerando uma dada margem de erro, realizar a previsão de um incidente de manutenção; no caso, podemos indicar o risco desse incidente passar de “baixa prioridade” para “alta prioridade”, possibilitando a tomada de um plano de ação, realocação de recursos, de orçamento, entre diversas outras formas de otimizar o atendimento dessa empresa, em relação a esses incidentes.

1.2 Objetivos

Em nosso cenário, existe a demanda de uma organização real, sediada em Sevilla, Espanha, para que possamos fazer a predição de mudança de prioridade sobre um incidente; de forma que uma ferramenta seja capaz de alertar, o quanto antes, o risco de certos incidentes tornarem-se mais complexos do que o previsto pelo atendimento inicial, de forma a reduzir custos através de um planejamento prévio e alocação de recursos otimizada.

O objetivo desse trabalho é, através do problema real de uma empresa, estudar e aplicar técnicas de classificação, lineares e não-lineares, utilizando a linguagem Python junto a biblioteca *scikit-learn*, que são padrões reconhecidos para estudos na área de ciência de dados. É importante que seja feita a utilização desses padrões, pois a biblioteca *scikit-learn* reúne diversas ferramentas eficientes, de simples utilização, para mineração e análise de dados. Também é favorável que essa ferramenta tenha código aberto e licença

BSD possibilitando, inclusive, seu uso comercial, além de uma comunidade ativa em portais de discussão.

Utilizando o problema como objeto de estudo, vamos analisar algumas abordagens possíveis de solução, apresentando, mensurando e comparando seus resultados. Nesse trabalho, vamos utilizar três métodos de aprendizagem: regressão logística (modelo linear), árvore de decisão (modelo não-linear) e *random forest* (*ensemble* de árvores de decisão, não-linear).

1.3 Organização do texto

O presente trabalho está estruturado em capítulos e, além desta introdução, será desenvolvido da seguinte forma:

- Capítulo II: Apresentação da demanda e do cenário de negócio, fundamentação matemática para as técnicas estatísticas que serão utilizadas e modelagem inicial do problema.
- Capítulo III: Visualização da base de dados de entrada, descrição das tecnologias utilizadas: softwares, bibliotecas Python e suas classes; modelagem do problema em Python, preparação dos dados e execução dos algoritmos.
- Capítulo IV: Análise qualitativa dos resultados: performance dos algoritmos, *accuracy* e validação cruzada (*k-fold*); visualização e interpretação gráfica dos resultados.
- Capítulo V: Considerações finais e sugestão de trabalhos futuros.

2 Cenário de Negócio e Fundamentação Teórica

Nesse capítulo, será apresentada a empresa e o cenário do problema apresentado pela mesma, que é objeto de estudo desse trabalho, sobre o qual vamos desenvolver nossa solução de aprendizado de máquina.

Também vamos detalhar como funcionam os métodos de aprendizagem que serão utilizados: regressão logística, árvore de decisão e *random forest*. Além disso, vamos explorar o conceito de variáveis dependentes e variáveis independentes, para que seja possível criar o modelo inicial.

2.1 Cenário da Empresa

Como apresentado nos objetivos desse trabalho (seção 1.2), o estudo é feito sobre um problema real de uma organização da Espanha. Os dados utilizados como insumo foram obtidos a partir do departamento de tecnologia da informação dessa empresa.

No cenário, temos que a organização possui uma área de suporte aos usuários, estruturando sua atuação por incidentes; ou seja, cada demanda de um cliente é um incidente exclusivo, que pode passar por diversas etapas, pessoas e estados. O processo é apresentado na Figura 1, abaixo. O processo foi obtido em [5], que representa o modelo no *framework ITIL*[®] 2:

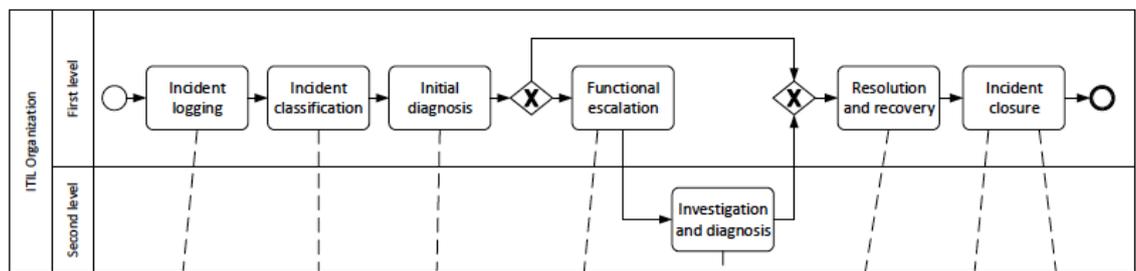


Figura 1 - Modelo de Processo de Gestão de Incidentes

Em um atendimento ótimo, o incidente é resolvido no primeiro nível, ou seja, a primeira pessoa a lidar com o problema do cliente é capaz de resolvê-lo satisfatoriamente, e aquele incidente é dado como encerrado.

Entretanto, existem ainda outros dois níveis para os times de suporte. No total, temos o primeiro nível como atendimento inicial ao usuário, o segundo nível como atendimento mais específico, no nível técnico-funcional e o terceiro nível como atendimento técnico especialista, que atua em incidentes de maior complexidade.

O problema da organização é que pode ocorrer que a priorização feita pelo primeiro nível seja equivocada, de forma que ocorra a alteração de priorização, enquanto o processo, em relação àquele incidente, já está em andamento. Isso causa consequências negativas para a gestão desses incidentes, principalmente em capacidade de atendimento, uma vez que não era prevista a alteração de prioridade.

2.2 Fundamentos de *Machine Learning*

De acordo com Mohri [6], o objetivo principal de um aprendiz é generalizar a partir de sua experiência. Nesse contexto, a generalização é a habilidade da máquina realizar previsões precisamente sobre novos dados, jamais processados pela máquina (novas experiências), após a experiência sobre a base de dados de treinamento. Esses exemplos de treinamento vêm de uma distribuição estatística desconhecida, e o aprendiz (no caso, a máquina) deve ser capaz de construir um modelo genérico sobre o espaço das ocorrências, de forma que seja capaz de realizar previsões suficientemente precisas sobre novas ocorrências.

Ainda que as tecnologias utilizadas na implementação possuam vasta documentação e materiais de apoio para cientistas de dados, é importante ter domínio sobre todas as possíveis técnicas de solução, ou ainda, testar todas essas técnicas para que seja possível comparar os diferentes resultados, em relação à eficiência e eficácia. Dessa forma, existirá a garantia de que estamos utilizando o melhor modelo para esse determinado problema.

Na seção seguinte, será apresentada a fundamentação matemática necessária para que possamos compreender cada método de aprendizagem utilizado nesse trabalho.

2.2.1 Métodos de Aprendizagem para Classificação

Diferentemente dos métodos de regressão, onde é feita a predição de uma variável numérica contínua, os métodos de classificação são utilizados para que se faça a predição de uma categoria. Suas aplicações são diversas, desde áreas como a medicina até tecnologias para jogos como o Microsoft Kinect [4].

Tabela 1 - Vantagens e Desvantagens de Modelos de Classificação

Modelo	Vantagens	Desvantagens
K-NN	Intuitivo, interpretável; treinamento rápido.	Necessidade de escolha de k vizinhos; estatisticamente incapaz de lidar com <i>features</i> de baixa relevância.
Regressão Logística	Abordagem probabilística; baixa variância; bons resultados sobre poucos dados.	Multicolinearidade; idealmente, requer uma grande quantidade de dados.
Naive Bayes	Eficiente em problemas não-lineares.	Alta necessidade de parametrizações; estatisticamente incapaz de lidar com <i>features</i> de baixa relevância.
Árvore de Decisão	Interpretável; sem necessidade de <i>feature scaling</i> ; adequado para variáveis categóricas.	<i>Overfitting</i> .
Random Forest	Excelente performance e precisão sobre a maioria dos problemas, lineares ou não-lineares; baixa variância.	Necessidade de escolha de n árvores; <i>overfitting</i> .

SVM	Não é sensível a <i>outliers</i> ou <i>overfitting</i> .	Incapaz de lidar com um número grande de <i>features</i> ; inapropriado para problemas não-lineares.
-----	--	--

Os modelos de classificação podem ser lineares, como a regressão logística e a SVM (*support vector machine*), ou não-lineares, como K-NN (*K-Nearest Neighbors*), *Kernel SVM* e *random forests*. Considerando questões didáticas e, também, levando em conta os algoritmos mais adequados para esse cenário (conforme será verificado na seção 2.2.3), as tarefas serão realizadas utilizando regressão logística, árvore de decisão e *random forest*, nesse trabalho.

2.2.1.1 Regressão Logística

A *logistic regression*, também conhecida como *logit regression*, *maximum-entropy classification (MaxEnt)* e *log-linear classifier*, na literatura, é um modelo linear onde as probabilidades de resultados de uma amostra são modeladas através de uma função logística.

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

Figura 2 - Equação da Função Logística

Onde:

- e: número de Euler;
- x_0 : valor de x no ponto médio da curva;
- L: valor máximo da curva;
- k: declividade da curva.

O gráfico da função segue o padrão da curva sigmoide. Na Figura 3, duas curvas possíveis são apresentadas.

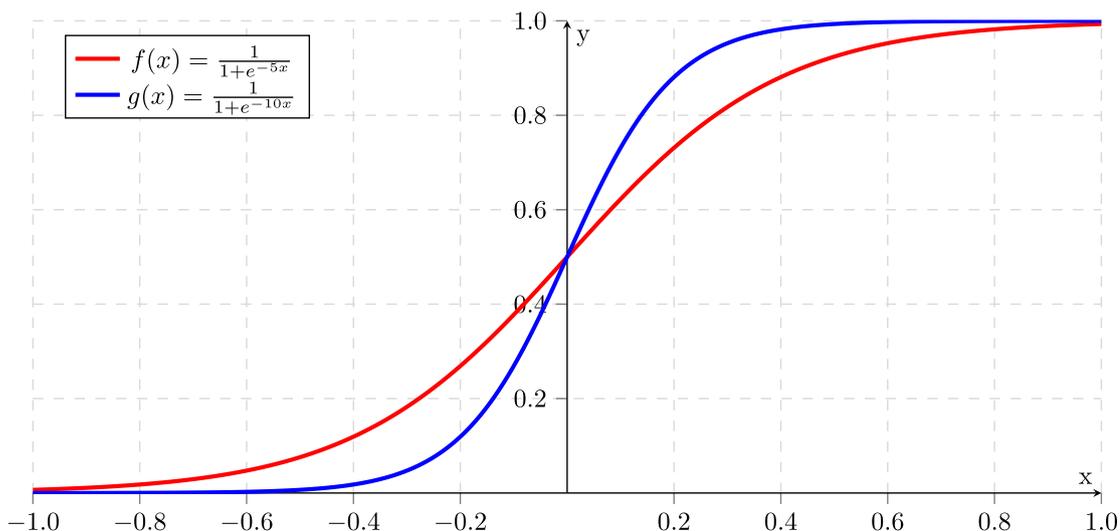


Figura 3 - Gráfico da Função Logística

Em termos de aprendizado de máquina, o que acontecerá, na prática, é que o classificador de regressão logística vai modelar a curva da melhor forma possível, com base nos dados de treinamento.

Assim, quando o modelo estiver ajustado, ele será testado contra dados específicos para teste (dados diferentes da etapa de treinamento). Então, teremos as previsões e podemos utilizar algumas métricas para calcular a precisão do modelo.

2.2.1.2 Árvore de Decisão

Na literatura, é muito comum a referência ao termo *CART (Classification and Regression Trees)*, introduzido por Breiman [7]. Essa nomenclatura serve como um “container” para os dois tipos distintos: árvores de classificação e árvores de regressão. Nesse trabalho, concentrar-nos-emos em árvores de decisão para classificação.

Árvore de Decisão, no contexto de algoritmos de classificação e aprendizado de máquina, é um modelo não-linear de aprendizagem, onde a predição da variável-alvo é feita através de regras de decisão simples, inferidas pelos dados de treinamento.

O modelo, através de complexas análises matemáticas, deve ser capaz de dividir o universo dos dados em subáreas, baseado em testes com valores de *features* (variáveis independentes). Esse processo de divisão é comumente chamado de *splitting*, e é realizado de forma recursiva; o processo só termina quando todas as subáreas conseguem ser bem definidas, de forma que todas as observações dentro de uma subárea possuam o mesmo valor de variável-alvo (variável dependente).

Na Figura 4, nota-se uma base de dados totalmente dividida (classificada), após o processo de *splitting*. No caso, existem apenas duas variáveis independentes, possibilitando que o gráfico pudesse ser representado no plano cartesiano, em duas dimensões. Em havendo mais variáveis independentes, o gráfico poderia ser um plano no espaço ou, inclusive, um cenário sem representação visual trivial.

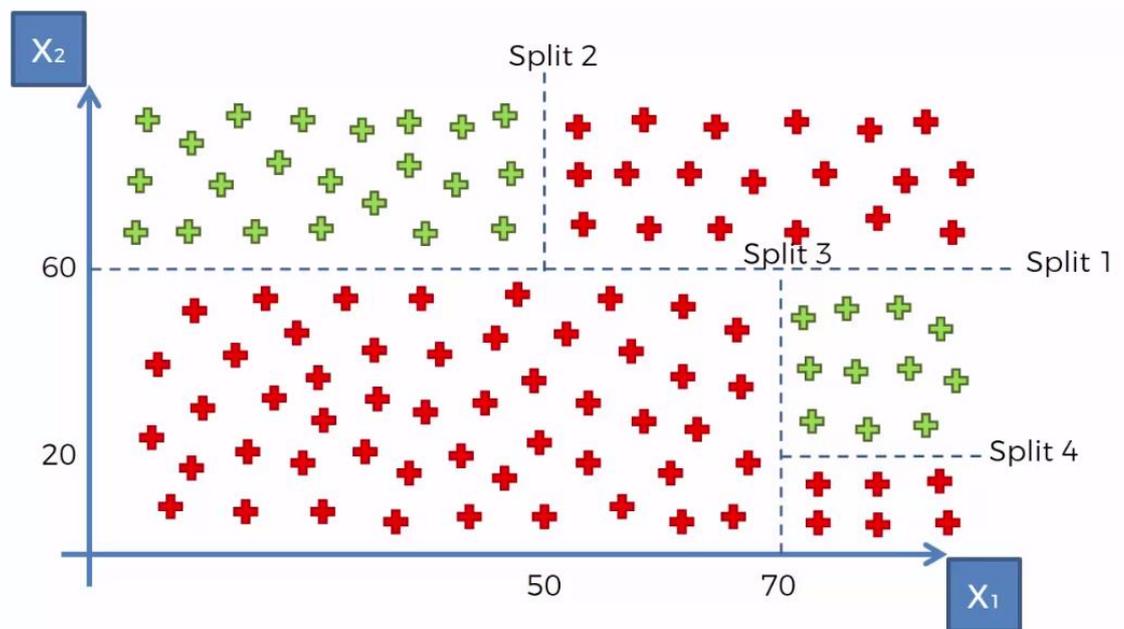


Figura 4 - Processo de splitting de uma Árvore de Decisão

Na Figura 5, temos a representação em árvore de decisão dessa classificação. Nota-se que as subáreas definidas são exatamente as folhas da árvore.

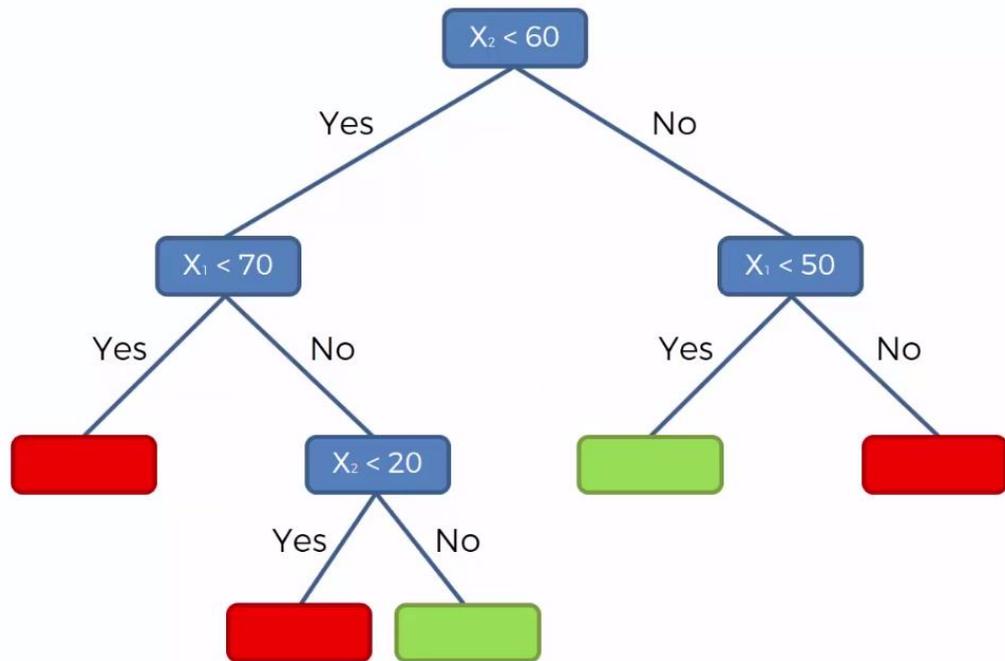


Figura 5 - Exemplo de uma Árvore de Decisão

Na Figura 6, vemos uma árvore de decisão mais complexa, próxima à realidade de *data science*, exportada do código Python através do método `tree.export_graphviz`, para a classificação de espécies de flores, utilizando dados como tamanho das pétalas e sépalas. Nesse exemplo, foi utilizado o *Iris flower dataset* [8].

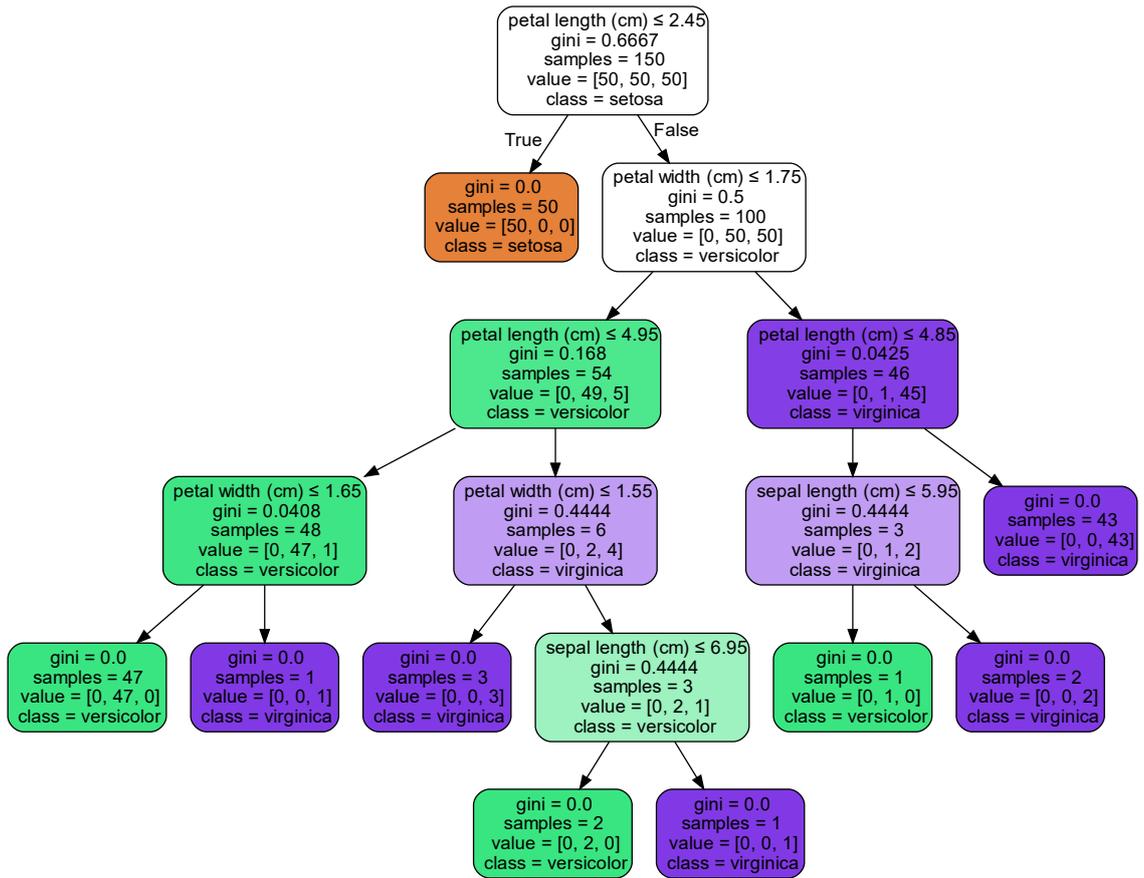


Figura 6 - Exemplo de uma Árvore de Decisão para o *dataset* iris

2.2.1.3 Random Forest

Para compreensão de *Random Forest*, é importante definir o termo *ensemble learning*, que é designado para aqueles métodos de aprendizagem que usam a combinação de múltiplos algoritmos de aprendizagem ao invés de usar um único para a predição, de forma a obter melhor performance preditiva. [9]

Random Forest é classificado como um *ensemble method* pois, como seu nome sugere, trata-se de uma floresta de árvores de decisão. Diferentemente da árvore de decisão simples, onde é definido o “melhor *split*” sobre todos os atributos, para classificação dos dados, *random forests* utilizam o “melhor *split*” sobre uma seleção randômica de atributos.

Para esse modelo, é definido um parâmetro de *estimators* – quantidade de árvores da floresta – onde cada árvore, utilizando essa seleção randômica, resulta em uma classificação (predição) sobre uma dada observação X. O resultado final do

algoritmo é dado pela moda estatística das categorias de todos os resultados, para fins de classificação, ou a média numérica, para fins de regressão.

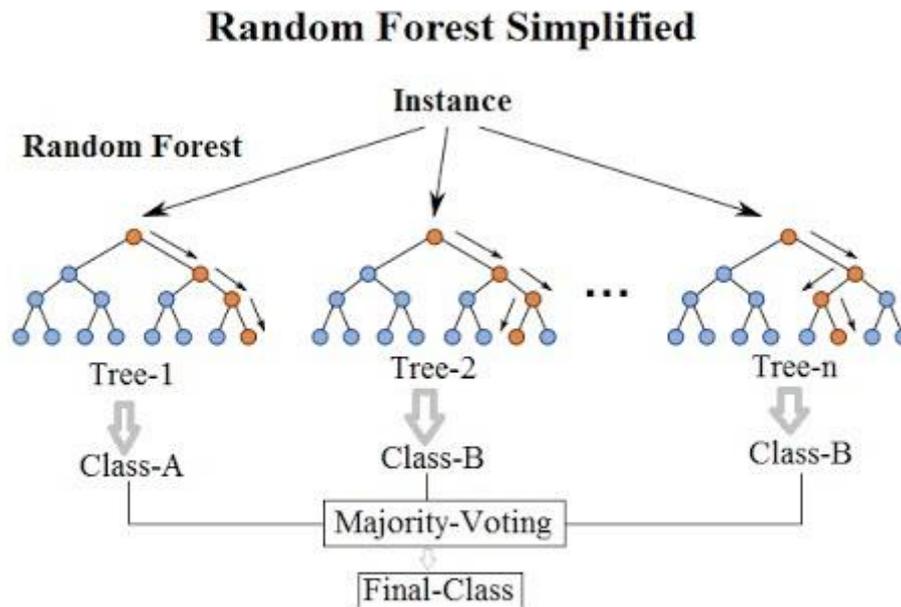


Figura 7 - Funcionamento de uma *Random Forest*

2.2.2 Variáveis Dependentes e Independentes

Em áreas como modelagem matemática e estatística, variáveis dependentes são exatamente aquelas cujos valores são dependentes de valores de outras variáveis, que são as variáveis independentes. Enquanto as variáveis independentes servem como entrada de dados, a variável dependente expressa o resultado, o produto de uma modelagem.

Essas variáveis podem ser de diferentes naturezas; podem ser valores numéricos contínuos, binários, categóricos, entre outros. Em nosso cenário, estaremos trabalhando apenas com valores categóricos. Também vamos notar, na seção 3.2.2.1, que os tipos de variáveis que estamos trabalhando influenciam na etapa de preparação dos dados.

Em se tratando de *machine learning*, as variáveis independentes também podem ser chamadas de *predictor variables* ou, pelo termo mais utilizado, *features*. A variável dependente, por sua vez, pode ser chamada de *predicted variable* ou *label*. Para a preparação do código Python, vamos chamar o conjunto de *features* por *feature matrix*,

e também chamaremos o conjunto de valores da variável dependente por *dependent vector*. Essas estruturas serão detalhadas na seção 3.2.2.2.

2.2.3 Aplicação para o Cenário de Negócio

Como apresenta a Tabela 1 (seção 2.2.1), cada modelo de classificação possui vantagens e desvantagens, que podemos entender como indicações e contraindicações de sua utilização, de acordo com um cenário específico.

Para esse trabalho, pode-se destacar as seguintes características, em relação à base de dados e ao problema que queremos resolver:

- Existe uma quantidade de *features* (atributos) relativamente alta, são 13 *features*, além da variável independente;
- Existe uma grande quantidade de registros (aproximadamente 175 mil);
- Estatisticamente, não é desejado perder nossos *outliers*; o evento de mudança de prioridade ocorre em uma baixa frequência, logo, não podemos utilizar um algoritmo que despreze esses valores.

No próximo capítulo, vamos considerar todas essas características para a construção do código Python e parametrização dos algoritmos que escolhemos utilizar: regressão logística, árvore de decisão e *random forest*. Também será feita a pré-visualização da base de dados e a definição das tecnologias utilizadas.

3 Aplicação de *Machine Learning*: Classificação

Neste capítulo, serão definidas todas as tecnologias que suportarão a modelagem e a execução dos algoritmos de aprendizado de máquina para o problema de classificação previamente descrito no cenário (seção 2.1).

No primeiro momento de execução, serão visualizados os dados da Base de Incidentes, nossa principal fonte de dados. Um dicionário de dados será apresentado de forma a auxiliar o entendimento sobre a base, possibilitando a modelagem correta das variáveis, para os modelos.

Para as tecnologias, teremos: a *IDE (Integrated Development Environment)* do projeto, o gerenciamento de pacotes e todas as bibliotecas Python – específicas para ciência de dados, matemática e plotagem de gráficos – utilizadas; além de conceituarmos as técnicas necessárias para garantir a execução correta dos algoritmos, como o pré-processamento de dados, o *train-test split* e o *feature scaling*.

3.1 Tecnologias do Ambiente Python

Dentro do universo de estudos sobre ciência de dados, as linguagens mais utilizadas são R e Python. O primeiro, por sua natureza estatística e uma diversidade de recursos matemáticos para os modelos e visualizações gráficas; já o segundo, por conta da facilidade de operação (curva de aprendizagem) e sua utilização generalizada em diversas áreas, como desenvolvimento *web*.

Nesse trabalho, será adotada a linguagem Python, por conta de sua sinergia com a ementa do curso de Sistemas de Informação, além das vantagens descritas acima. As tecnologias Python utilizadas serão definidas, nessa seção.

3.1.1 Plataforma Anaconda e Spyder IDE

Anaconda 5.0.1 é uma das plataformas mais famosas do mundo para *data science* em Python, possuindo mais de 1.000 pacotes específicos da área e mais de 4,5 milhões de usuários. Algumas das aplicações pré-instaladas são: *Jupyter Notebook*, *Spyder* e *RStudio*.

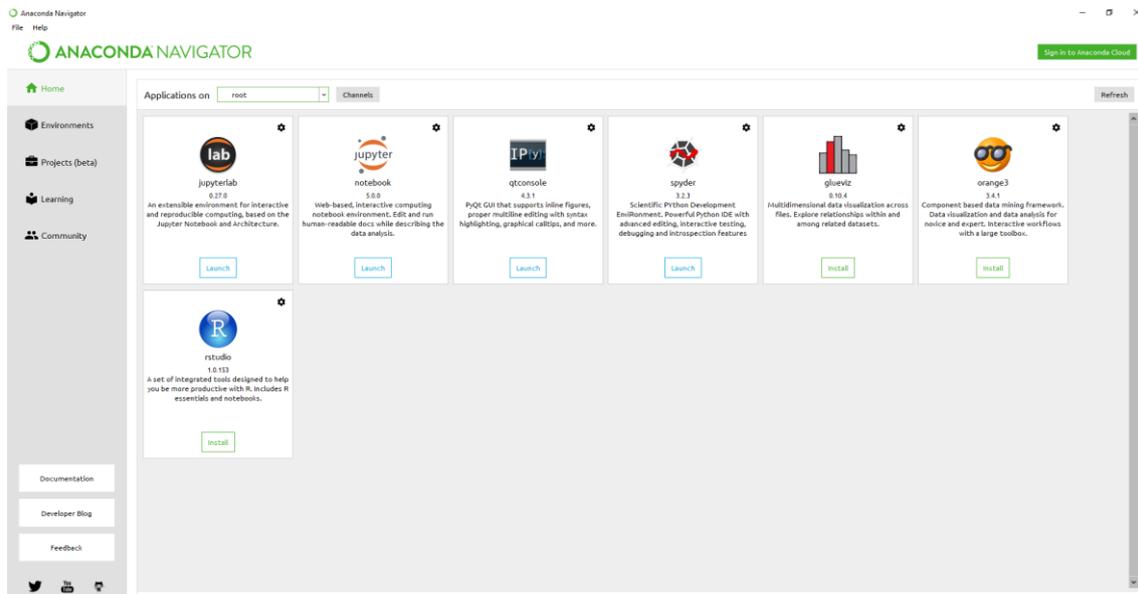


Figura 8 - Anaconda Navigator

Uma de suas aplicações chama-se Spyder 3 IDE, que é um ambiente de desenvolvimento Python com diversos recursos, como:

- Editor de código com documentação de funções e classes e *code completion*;
- Console interativo IPython [10] com suporte à depuração do código executado e integração com a biblioteca matplotlib;
- Explorador de variáveis para análise dinâmica de valores de execução;
- Explorador de arquivos.

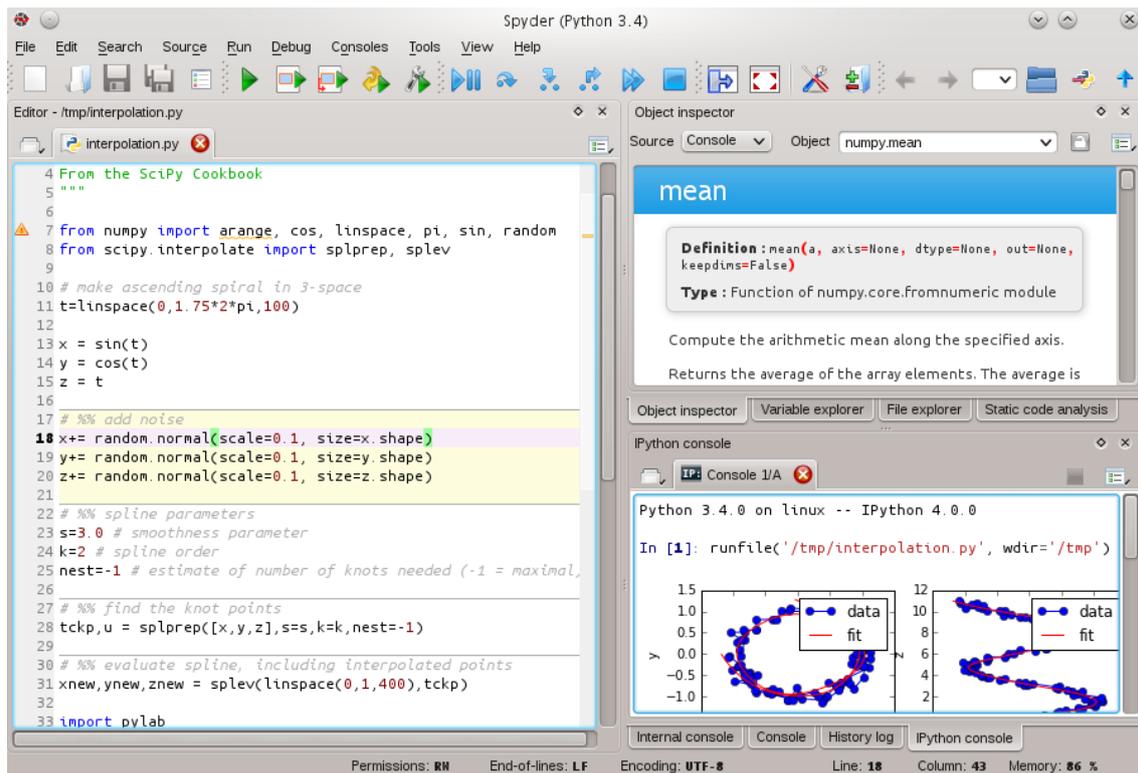


Figura 9 - Spyder IDE

3.1.2 Bibliotecas

A maioria das bibliotecas utilizadas já vem pré-instaladas no ambiente Anaconda. Entretanto, pode ser necessário instalar novas bibliotecas, utilizando o Terminal ou o Anaconda Prompt, através do seguinte comando:

> *conda install 'pacote'*

Em um ambiente Python padrão, pode ser utilizado o comando pip:

> *pip install 'pacote'*

```
Administrador: C:\WINDOWS\system32\cmd.exe
(C:\Users\lucas.LX31\Anaconda3) C:\Users\lucas.LX31\Documents>conda install reportlab
Fetching package metadata .....
Solving package specifications: .

Package plan for installation in environment C:\Users\lucas.LX31\Anaconda3:

The following NEW packages will be INSTALLED:

  reportlab: 3.4.0-py36_0

The following packages will be UPDATED:

  conda:      4.3.27-py36hcbae3bd_0 --> 4.3.30-py36h7e176b0_0

Proceed ([y]/n)? y

reportlab-3.4. 100% |#####| Time: 0:00:01  1.74 MB/s
conda-4.3.30-p 100% |#####| Time: 0:00:00  4.49 MB/s
```

Figura 10 - Instalação de Pacote por Prompt de Comando

3.1.2.1 scikit-learn

scikit-learn é uma biblioteca de software para *machine learning* destinada a linguagem Python [11]. Ela possui diversos algoritmos para classificação, regressão e clustering, como *support vector machines*, *random forests*, *gradient boosting* e *k-means*.

Foi criada em 2007, por David Cournapeau, como um projeto do *Google Summer of Code*. Em 2010, sob novas lideranças, teve sua primeira divulgação pública e está sob desenvolvimento ativo até então (dezembro de 2017). Sua última versão estável é a versão 0.19.0, disponibilizada em agosto de 2017.

3.1.2.2 pandas

pandas é uma biblioteca de alta performance que provê estruturas de dados e ferramentas intuitivas de análise e manipulação de dados para a linguagem Python. [12] Criada por Wes McKinney - cientista de dados e estatístico americano – hoje, o projeto conta com o suporte da comunidade e com apoio institucional de organizações como a *Anaconda, Inc.* e o *Paris-Saclay Center for Data Science*.

Alguns de seus principais recursos são:

- Tratamento de dados nulos;
- Redimensionamento de *DataFrames* e objetos de maior dimensão;
- Alinhamento automático entre os dados e os cabeçalhos;
- Agrupamento, *subsetting*, *reshaping* e *pivoting* de dados;

- Diversos formatos de entrada, como Excel, CSV, bancos de dados, entre outros.

3.1.2.3 numpy

NumPy é uma biblioteca fundamental para computação científica na linguagem Python. Ela suporta operações para vetores e matrizes multidimensionais, juntamente a uma coleção de funções matemáticas de alta abstração. [13]

Previamente chamada de Numeric, a biblioteca NumPy foi criada por Travis Oliphant, em 2006, no momento que unificou os recursos da biblioteca Numarray com os recursos da Numeric, unificando a comunidade em torno de um único pacote para operações vetoriais.

Seus principais recursos são:

- Objetos e métodos para vetores N-dimensionais;
- Ferramentas úteis para álgebra linear, transformações de Fourier e manipulação de números randômicos.

3.2 Execução das Máquinas

3.2.1 Base de Dados de Incidentes

A Base de Dados de Incidentes é a principal fonte de dados do projeto. Ela possui aproximadamente 175 mil registros, onde cada registro representa um incidente para o setor de atendimento da organização. A base está armazenada em um arquivo CSV, com dados históricos.

A base possui 30 colunas, onde podemos dividir em 2 eventos sequenciais. Ou seja, para cada incidente, existem 15 colunas referentes ao primeiro evento (por exemplo, o atendimento inicial de um cliente) e mais 15 colunas referentes ao segundo evento (o evento subsequente ao primeiro. Por exemplo, encaminhamento do incidente para o departamento de TI).

Para o cenário desse trabalho, entretanto, não será necessário trabalhar com todas essas colunas. Uma vez que se deseja treinar o modelo com base em dados do primeiro evento, para a predição de prioridade do segundo evento, precisamos, somente, do atributo “prioridade” do segundo evento, além de todos os atributos do primeiro evento.

Tabela 2 - Dicionário de Dados da Base de Incidentes

ATRIBUTO	DESCRIÇÃO	TIPO	NULO
AUTOR	Usuário da atividade	Texto	NÃO
NODO	Nome do Nó qual o Centro pertence	Texto	NÃO
RESOLUTOR	Organização funcional do delegado à atividade	Texto	NÃO
TIPOLOGIA	Categoria descritiva do tipo de incidente	Texto	NÃO
ASIGNATARIO	Usuário a quem a atividade é atribuída	Texto	NÃO
PRIORIDAD	Indicador do nível de prioridade da atividade	Texto	NÃO
GRUPOAUTOR	Organização funcional do autor	Texto	SIM
ESTADO	Estado da atividade	Texto	NÃO
CENTRO	Nome do Centro relacionado ao incidente	Texto	NÃO
ORIGEN	Nome do canal de entrada da informação no sistema	Texto	NÃO
TYPEORG	Tipo de atividade	Texto	NÃO

RECURSO	Nome do recurso afetado pelo incidente	Texto	SIM
MOTIVOCIERRE	Descrição da razão para fechamento do incidente	Texto	SIM
PRIORIDAD2	Indicador do nível de prioridade da atividade, na segunda instância (sequência do incidente)	Texto	NÃO

3.2.2 Etapas da Execução Python

Nessa seção, será feita a apresentação do algoritmo em Python, com demonstração do código-fonte e explicações sobre cada etapa de execução.

3.2.2.1 Pré-processamento de dados

Para a execução dos algoritmos de *machine learning*, devemos garantir que os dados estejam no formato correto, através da realização de duas etapas: o tratamento de dados ausentes (dados nulos) e a codificação de variáveis categóricas. Se esses procedimentos não forem seguidos, os algoritmos informarão resultados errados, ou ainda, na maioria das vezes, não conseguirão iniciar sua execução.

Como primeiro passo, é feita a importação das bibliotecas que serão utilizadas no algoritmo (todas as bibliotecas foram previamente apresentadas, na seção 3.1.2), além da importação da base de dados para a memória.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4
5 # Importing the dataset
6 dataset = pd.read_csv('SAS_K06.csv', encoding='latin-1', sep=';',
7                       dtype={'RECURSO':str})
8 X = dataset.iloc[:, :-1].values
9 y = dataset.iloc[:, 13].values

```

Figura 11 - Código para Importação da Base CSV

Sobre o método `read_csv`: foi necessário que alguns argumentos fossem explicitados; o `encoding = 'latin-1'` (informalmente, ISO/IEC 8859-1), pois a base de dados está em espanhol, o `sep = ';'` para definir o separador que delimita as colunas da base e `dtype = {'RECURSO':str}` para que o algoritmo compreenda a coluna 'RECURSO' usando o tipo de dados *string*. O trabalho de identificação do tipo de dado é feito automaticamente pela biblioteca pandas; entretanto, podem haver problemas se uma mesma coluna possui tipos de dado distintos.

3.2.2.1.1 Tratamento de Dados Ausentes

Em algumas colunas da base de dados utilizadas, e de acordo com o dicionário de dados (apresentado na seção 3.2.1), existem alguns dados que não possuem valor associado. Na Figura 12, utilizando o método *describe*, podemos ter um resumo da base de dados importada. Na Figura 13, através do método *isnull* é feita uma verificação de valores nulos, para cada coluna.

```

In [6]: print(dataset.describe())
count      AUTOR      NODO      RESOLUTOR      TIPOLOGIA \
unique      2325      9      77      49
top      'P U S' SEVILLA RESOLUTOR-PUESTO-USUARIO Incidencia.Hardware
freq      110712  33523      126631      97889

count      ASIGNATARIO PRIORIDAD      GRUPOAUTOR ESTADO \
unique      549      3      78      5
top      'PUESTO USUARIO S' P3 RESOLUTOR-PUESTO-USUARIO Fijada
freq      126627  170091      110712  106159

count      CENTRO      ORIGEN TYPEORG      RECURSO \
unique      1888      4      9      28276
top      'H. G. BASICO BAZA' TELEFONO RE 1,06E+11
freq      7430      96963  106012      35

count      MOTIVOCIERRE PRIORIDAD2
unique      162      3
top      'Recurso.Configuracion logica' P3
freq      1392      169639

```

Figura 12 - Resumo da Base de Dados

```

In [10]: print(dataset.isnull().sum())
AUTOR      0
NODO      0
RESOLUTOR  0
TIPOLOGIA  0
ASIGNATARIO  0
PRIORIDAD  0
GRUPOAUTOR  13285
ESTADO      0
CENTRO      0
ORIGEN      0
TYPEORG      0
RECURSO      146380
MOTIVOCIERRE  167325
PRIORIDAD2  0
dtype: int64

```

Figura 13 - Contagem de Valores Nulos da Base de Dados

Como previsto pelo dicionário de dados (seção 3.2.1), existem valores nulos para as colunas ‘GRUPOAUTOR’, ‘RECURSO’ e ‘MOTIVOCIERRE’. Analisando também a Figura 13, verificamos que ‘RECURSO’ possui, praticamente, um valor para cada incidente, quando não está vazio (está vazio em, aproximadamente, 84% dos incidentes), e que ‘MOTIVOCIERRE’ está vazio em 96% dos casos. Logo, a decisão tomada é de remover essas duas colunas, uma vez que não terão utilidade para o modelo podendo, inclusive, orientar os classificadores ao erro.

```

In [19]: dataset = dataset.drop(['RECURSO', 'MOTIVOCIERRE'], axis=1)

In [20]: print(dataset.isnull().sum())
AUTOR                0
NODO                 0
RESOLUTOR            0
TIPOLOGIA            0
ASIGNATARIO         0
PRIORIDAD            0
GRUPOAUTOR          13285
ESTADO               0
CENTRO               0
ORIGEN               0
TYPEORG              0
PRIORIDAD2           0
dtype: int64

```

Figura 14 - Contagem de Valores Nulos após Remoção de Colunas

Na coluna 'GRUPOAUTOR', uma opção comum seria remover essas linhas da base, onde a coluna possui dados vazios. Porém, quando falamos de aprendizado de máquina e ciência de dados, também pensando no cenário do projeto, pode-se perder informações preciosas como alguns *outliers*, por exemplo, se utilizarmos essa medida. Então, o tratamento para essas linhas (incidentes da base de dados) será substituir os valores indesejáveis (nulos) pela moda de todos os valores, naquela coluna. Dessa forma, estatisticamente, não haverá perda de nenhum registro, o que é muito importante para o aprendizado de máquina.

```

In [37]: dataset['GRUPOAUTOR'].mode()[0]
Out[37]: 'RESOLUTOR-PUESTO-USUARIO'
In [38]: dataset['GRUPOAUTOR'].fillna(dataset['GRUPOAUTOR'].mode()[0], inplace=True)

In [39]: print(dataset.isnull().sum())
AUTOR          0
NODO           0
RESOLUTOR      0
TIPOLOGIA      0
ASIGNATARIO    0
PRIORIDAD      0
GRUPOAUTOR     0
ESTADO         0
CENTRO         0
ORIGEN         0
TYPEORG        0
PRIORIDAD2     0
dtype: int64

In [40]: print(dataset.describe())
count      AUTOR      NODO      RESOLUTOR      TIPOLOGIA  \
unique     2325      9      77      49
top        'P U S'  SEVILLA  RESOLUTOR-PUESTO-USUARIO  Incidencia.Hardware
freq       110712   33523      126631      97889

count      ASIGNATARIO  PRIORIDAD      GRUPOAUTOR  ESTADO  \
unique     549      3      78      5
top        'PUESTO USUARIO S'      P3  RESOLUTOR-PUESTO-USUARIO  Fijada
freq       126627   170091      123997   106159

count      CENTRO  ORIGEN  TYPEORG  PRIORIDAD2
unique     1888      4      9      3
top        'H. G. BASICO BAZA'  TELEFONO      RE      P3
freq       7430      96963  106012   169639

```

Figura 15 - Resumo da Base de Dados após Tratamento de Valores Nulos

3.2.2.1.2 Codificação de Variáveis Categóricas

Na base de dados utilizada, todas as colunas são do tipo texto. Acima disso, todas as colunas podem ser chamadas de variáveis categóricas, pois assumem valores dentro de um domínio conhecido dos valores possíveis.

Para os algoritmos, não é possível executar as instruções utilizando dados de texto; são necessários números reais. Logo, utilizamos a classe *LabelEncoder*, específica para realizar a codificação de variáveis categóricas em variáveis numéricas, possibilitando a execução dos algoritmos.

```

23 # Encoding categorical data
24 from sklearn.preprocessing import LabelEncoder
25 labelencoder = LabelEncoder()
26 for x in range(0, 11):
27     X[:, x] = labelencoder.fit_transform(X[:, x])
28 y = labelencoder.fit_transform(y)

```

Figura 16 - Implementação para Codificação Numérica de Variáveis Categóricas

Index	AUTOR	NODO	RESOLUTOR	TIPOLOGIA	ASIGNATARIO	PRIORIDAD	GRUPOAUTOR	ESTADO	CENTRO	ORIGEN	TYPEORG
0	114	1	6	12	17	2	5	0	567	3	3
1	833	3	6	12	223	2	5	0	533	3	3
2	833	3	57	12	489	2	5	0	533	3	8
3	306	5	57	12	489	2	61	0	531	1	8
4	1872	5	6	12	462	2	5	0	1813	3	3
5	1872	5	52	12	127	2	5	0	1813	3	8
6	306	5	57	12	489	2	61	0	531	1	8
7	214	0	6	12	25	2	5	0	545	3	3
8	214	0	6	12	25	2	5	0	545	3	2
9	214	0	57	12	489	2	5	0	545	3	8
10	1893	8	39	21	446	2	32	0	999	1	7
11	1893	7	5	21	65	2	32	0	999	1	8
12	1679	7	6	21	414	2	5	0	999	1	8
13	1679	7	7	21	504	2	5	0	999	1	8

Figura 17 - Base de Dados após Codificação Numérica

Entretanto, essa codificação impõe uma “ordinalidade” que não se aplica para o cenário. Por exemplo, se temos os valores do atributo ‘ORIGEN’ “TELEFONO”, “EMAIL” e “INTRANET” codificados, respectivamente como ‘0’, ‘1’ e ‘2’, é como se “TELEFONO” fosse menor que “EMAIL”, ou “INTRANET” fosse o dobro de “EMAIL”. Essas afirmações não fazem sentido, são incorretas, mas vale ressaltar que não há impacto sobre os algoritmos de árvore de decisão e *random forest*.

Quando existe essa situação, onde não queremos a relação de ordem que os números impõem, utilizamos a classe *OneHotEncoder*. Essa classe é responsável por transformar uma matriz de números inteiros (que denotam os valores de variáveis categóricas/discretas) em uma matriz esparsa, onde cada coluna representa um valor

binário, sobre a presença ou ausência daquela categoria. Na Figura 18, temos um exemplo para uma categoria “Nome”.

Categorical Feature		f1	f2	f3	f4	f5	f6	f7	f8	f9	f10
Louise	=>	1	0	0	0	0	0	0	0	0	0
Gabriel	=>	0	1	0	0	0	0	0	0	0	0
Emma	=>	0	0	1	0	0	0	0	0	0	0
Adam	=>	0	0	0	1	0	0	0	0	0	0
Alice	=>	0	0	0	0	1	0	0	0	0	0
Raphael	=>	0	0	0	0	0	1	0	0	0	0
Chloe	=>	0	0	0	0	0	0	1	0	0	0
Louis	=>	0	0	0	0	0	0	0	1	0	0
Jeanne	=>	0	0	0	0	0	0	0	0	1	0
Arthur	=>	0	0	0	0	0	0	0	0	0	1

Figura 18 - Exemplo de Codificação *one-hot*

A codificação *one-hot* é obrigatória para modelos lineares e algumas SVM (*support vector machines*). Logo, precisaremos dessa etapa para a execução da regressão logística. Os algoritmos de árvore de decisão e *random forest* já possuem tratamento específico para variáveis categóricas, de forma que a codificação *one-hot* não seja necessária.

Logo, realizando esse procedimento para todas as *features*, chegamos a um novo número de features, muito superior ao inicial: 4985 atributos. A complexidade de cálculo e o tempo de execução do algoritmo para a regressão logística serão muito superiores aos dos outros algoritmos, devido a essas múltiplas dimensões.

3.2.2.2 *Feature Matrix* e Vetor Dependente

De acordo com o exposto na seção 2.2.2, as variáveis independentes são aquelas sobre as quais os algoritmos vão estabelecer uma função, ou seja, uma relação entre todas essas variáveis para que se tenha a predição da variável dependente.

Logo, teremos uma estrutura composta por 13 colunas (variáveis independentes) e todas as linhas da base de dados, chamada de *feature matrix*; ao passo que também precisamos criar uma estrutura composta por 1 coluna (variável dependente: PRIORIDAD2) e todas as linhas, chamada de vetor dependente.

O trabalho, em termos algorítmicos, está resumido em encontrar uma função sobre a feature matrix para que tenhamos resultado igual ao vetor dependente.

3.2.2.3 Train-Test Split

Uma vez que temos a base tratada contra dados nulos e dados categóricos, dividida nas estruturas corretas (*feature matrix* e vetor dependente), é possível realizar a etapa de divisão dos dados, onde parte da base de dados será utilizada pra o treinamento da máquina e a outra parte será utilizada pra testar a máquina a partir de dados diferentes. Por isso, chama-se de *train-test split*.

É uma etapa fundamental para qualquer algoritmo de *machine learning*. Por padrão, essa divisão assume uma taxa entre 0,2 e 0,25 para os dados de teste, enquanto todo o restante é usado para o treinamento do algoritmo. Ou seja, se utilizarmos o parâmetro *test_size = 0,2*, em uma base de dados com 1.000 registros, automaticamente, o tamanho da base de treinamento será de 800 registros, sendo: 800 registros da *feature matrix* e os 800 valores correspondentes a esses registros no vetor dependente. Na Figura 19, é apresentada a implementação para a realização dessa divisão de dados, utilizando *test_size = 0,25*.

```
30 # Splitting the dataset into the Training set and Test set
31 from sklearn.model_selection import train_test_split
32 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

Figura 19 - Código para Divisão de Dados de Teste e Treinamento

Um erro comum que pode ocorrer nessa etapa de codificação é o chamado *overfitting*. Esse problema ocorre quando o modelo é demasiadamente treinado/orientado aos dados de treinamento, de forma que é incapaz de realizar novas predições com precisão. Uma forma de reduzir o *overfitting*, de acordo com Cawley [14], é aplicar a validação cruzada; essa etapa será realizada na seção 4.2.2.

3.2.2.4 Feature Scaling

Feature Scaling é uma técnica necessária para alguns algoritmos, enquanto outros já realizam o *scaling* automaticamente, identificando os dados de entrada que estão sendo utilizados. A regressão logística, por exemplo, utiliza a fórmula da distância

euclidiana (Figura 20) para realizar as predições; não seria correto calcular a distância entre pontos usando uma coordenada de escala 0 até 10, enquanto uma outra coordenada poderia adotar a escala 0 até 1000.

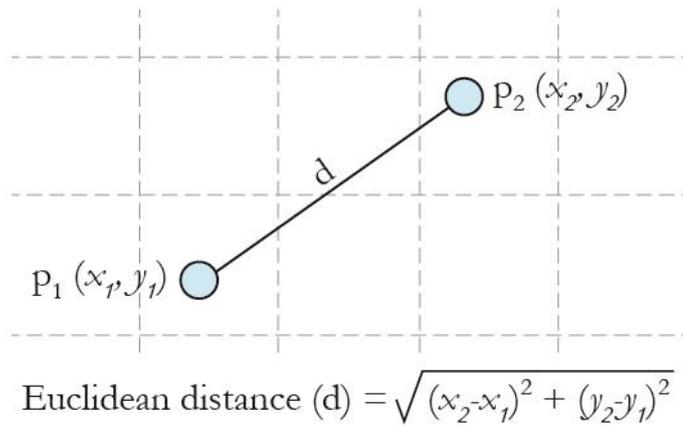


Figura 20 - Fórmula da Distância Euclidiana

Em síntese, o procedimento é a normalização desses valores numéricos, sem perda no poder comparativo, para que alguns algoritmos não cometam erros durante a etapa de *fitting* (adequação do modelo aos dados).

	0	1	2	3	4	5	6	7	8	9	10
0	0.458195	0.797974	0.540247	-0.382358	0.486796	0.160549	0.51372	0.784445	-0.255605	0.817531	-0.107956
1	0.458195	-1.23218	0.540247	-0.569822	0.486796	0.160549	0.51372	0.784445	-0.303929	-0.351563	-0.107956
2	-0.930478	1.20401	-0.159415	-0.382358	-1.0934	0.160549	0.51372	-1.25982	-0.191172	0.817531	1.8646
3	-1.28534	1.20401	-0.159415	-0.382358	-1.99312	0.160549	0.136427	-1.25982	0.337716	0.817531	-1.42299
4	0.458195	-1.23218	0.540247	-0.382358	0.486796	0.160549	0.51372	0.784445	-0.325407	-0.351563	-0.107956
5	-2.41351	-1.63821	0.540247	-0.382358	0.486796	0.160549	-2.2531	-1.25982	-0.258289	0.817531	2.52212
6	-1.40636	-0.826152	-2.05262	-0.382358	-1.3051	0.160549	-2.21118	-1.25982	-0.322722	0.817531	2.52212
7	0.458195	-1.63821	0.540247	2.05468	0.486796	0.160549	0.51372	-1.25982	-0.330777	0.817531	1.8646
8	-0.118197	-1.23218	0.540247	1.30482	0.486796	0.160549	-2.21118	-1.25982	-0.269028	0.817531	2.52212
9	0.458195	0.391942	0.540247	-0.382358	0.486796	0.160549	0.51372	0.784445	-0.282452	0.817531	-0.107956
10	1.08382	1.61004	-0.694451	-0.382358	-2.76432	0.160549	-1.0793	-1.25982	0.888082	-1.52066	-0.765474
11	0.458195	0.391942	0.540247	1.30482	0.486796	0.160549	0.51372	0.784445	-0.311984	0.817531	-0.107956
12	0.458195	1.20401	0.540247	-0.382358	0.486796	0.160549	0.51372	0.784445	-1.53353	0.817531	-0.107956

Figura 21 - Dados de Treinamento após Feature Scaling

3.2.2.5 Execução dos Classificadores

Nessa etapa, é onde o aprendizado de máquina efetivamente acontece. Os classificadores, representados pela variável *classifier*, no código-fonte, são as *machines*; enquanto o *learning* é o método que alimenta essas máquinas com dados de treinamento.

Por conta disso, usamos o termo "aprendizado de máquina"; as máquinas vão aprendendo sobre o comportamento daqueles dados, cada vez mais, enquanto recebem registro após registro, e vão construindo uma relação padrão, nos moldes daquele classificador específico. No presente cenário, os classificadores são baseados em regressão logística, árvore de decisão e *random forest*.

Nas Figuras 22, 23 e 24, vemos as implementações das classes da biblioteca *scikit-learn* para os três classificadores. Para todos os casos, temos a seguinte situação sobre as variáveis:

- **X_train:** *feature matrix* sobre os dados de treinamento. Matriz com todas as variáveis independentes;
- **y_train:** vetor de variáveis dependentes sobre os dados de treinamento;
- **X_test:** *feature matrix* sobre os dados de teste. Matriz com todas as variáveis independentes;
- **y_test:** vetor de variáveis dependentes sobre os dados de teste. Esses valores estão armazenados, mas não são utilizados para o modelo; são utilizados, somente, para verificação de precisão, ao fim dos cálculos;
- **y_pred:** valores de predição sobre os dados de teste.

```
20 # Fitting Logistic Regression to the Training set
21 from sklearn.linear_model import LogisticRegression
22 classifier = LogisticRegression(random_state = 0)
23 classifier.fit(X_train, y_train)
24
25 # Predicting the Test set results
26 y_pred = classifier.predict(X_test)
```

Figura 22 - Código para Classificação por Regressão Logística

Na figura acima, observa-se que importamos a classe *LogisticRegression*, da biblioteca de modelos lineares *sklearn.linear_model* e a variável *classifier* recebe uma

instância da classe. O método `fit` será responsável por adequar a curva sigmoide (como visto na seção 2.2.1.1), da regressão logística, aos dados de treinamento.

```
20 # Fitting Decision Tree Classification to the Training set
21 from sklearn.tree import DecisionTreeClassifier
22 classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
23 classifier.fit(X_train, y_train)
24
25 # Predicting the Test set results
26 y_pred = classifier.predict(X_test)
```

Figura 23 - Código para Classificação por Árvore de Decisão

Na figura acima, observa-se que importamos a classe `DecisionTreeClassifier`, da biblioteca de árvores `sklearn.tree` e a variável `classifier` recebe uma instância da classe. O método `fit` será responsável pelo *splitting* dos dados até que as folhas sejam alcançadas (como visto na seção 2.2.1.2).

O parâmetro `criterion` especifica o critério utilizado para medir a qualidade de cada divisão (*split*) da árvore. As opções são ‘`gini`’ para a utilização de impureza Gini e ‘`entropy`’ para “ganho de informação” (também chamado de Divergência de Kullback-Leibler [15]).

```
32 # Fitting Random Forest Classification to the Training set
33 from sklearn.ensemble import RandomForestClassifier
34 classifier = RandomForestClassifier(n_estimators = 100, criterion = 'entropy',
35                                  random_state = 0)
36 classifier.fit(X_train, y_train)
37
38 # Predicting the Test set results
39 y_pred = classifier.predict(X_test)
```

Figura 24 - Código para Classificação por *Random Forest*

Na figura acima, observa-se que importamos a classe `RandomForestClassifier`, da biblioteca de métodos `ensemble sklearn.ensemble` e a variável `classifier` recebe uma instância da classe. O método `fit` será responsável pelo *splitting* dos dados até que as folhas sejam alcançadas, em cada árvore, e ao final da execução de todas as árvores de decisão, a moda estatística entre as previsões será o resultado final da previsão *random forest* (como visto na seção 2.2.1.3), para cada registro. O parâmetro `n_estimators` especifica o número de árvores da floresta.

Para todos os três casos, o método *predict* é responsável por aplicar os dados de teste contra o modelo obtido após o treinamento. Os resultados das previsões são armazenados na variável *y_pred*; esses valores serão apresentados, na próxima seção.

4 Resultados dos Algoritmos de Análise Preditiva

Após as etapas de fundamentação, preparação e execução, então, esse capítulo apresentará resultados e análises sobre os resultados das previsões calculadas, a validação desses valores através de validação cruzada e representações sobre os resultados, como a árvore de decisão resultante e a *confusion matrix*, por exemplo.

Para a regressão logística, não vamos contar com uma representação gráfica do resultado, pois a codificação *one-hot* dos dados exigiu a criação de mais de 4000 *features* o que, matematicamente, significa que o problema é representado em muitas dimensões. Quando existe esse cenário, superior a 3 dimensões, a interpretação visual dos dados não é possível, pois trata-se de um hiperplano. Logo, os resultados da regressão logística estão restritos a *confusion matrix* (apresentada na seção 4.1.1).

4.1 Apresentação dos Resultados

4.1.1 *Confusion Matrix*

No campo de aprendizado de máquina e, especificamente, em classificação estatística, *confusion matrix* (ou *error matrix*; matriz de confusão, matriz de erro) é uma tabela específica que permite a visualização de resultados sobre os algoritmos de classificação, facilitando a interpretação sobre possíveis “confusões” que o computador tenha feito sobre as classes em questão [16].

Cada linha da tabela representa as instâncias da classe “verdadeira” (classificação correta), enquanto cada coluna representa as instâncias da classe prevista (classificação do algoritmo).

Em termos algébricos, considerando uma matriz $A[i][j]$, todas as previsões corretas estão representadas em elementos de posição $i = j$; ou seja, elementos da diagonal principal da matriz. Os demais elementos representam falsos positivos.

Após a execução dos três algoritmos, os classificadores já foram treinados sobre os dados de treinamento, e as previsões para os dados de teste também já foram feitas.

É muito importante verificar, quantitativamente, qual foi a taxa de acerto de cada algoritmo. Para isso, vamos utilizar o método mais simples de verificação, que é a análise sobre a *confusion matrix*.

Na Figura 25, temos a *confusion matrix* do algoritmo de regressão logística :



Figura 25 - Confusion Matrix da Classificação por Regressão Logística

Na Figura 26, temos a *confusion matrix* do algoritmo de árvore de decisão:

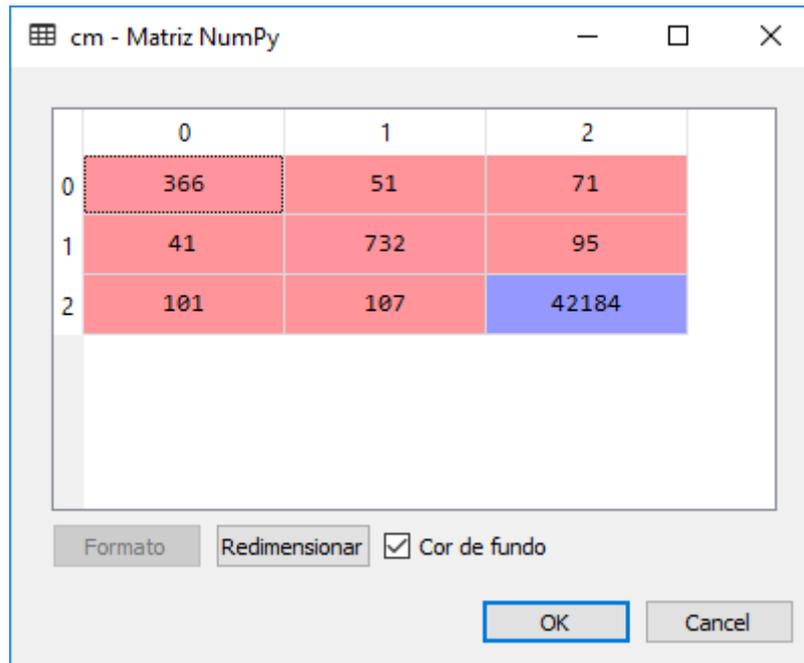


Figura 26 - Confusion Matrix da Classificação por Árvore de Decisão

Na Figura 27, temos a *confusion matrix* do algoritmo de *random forest*:

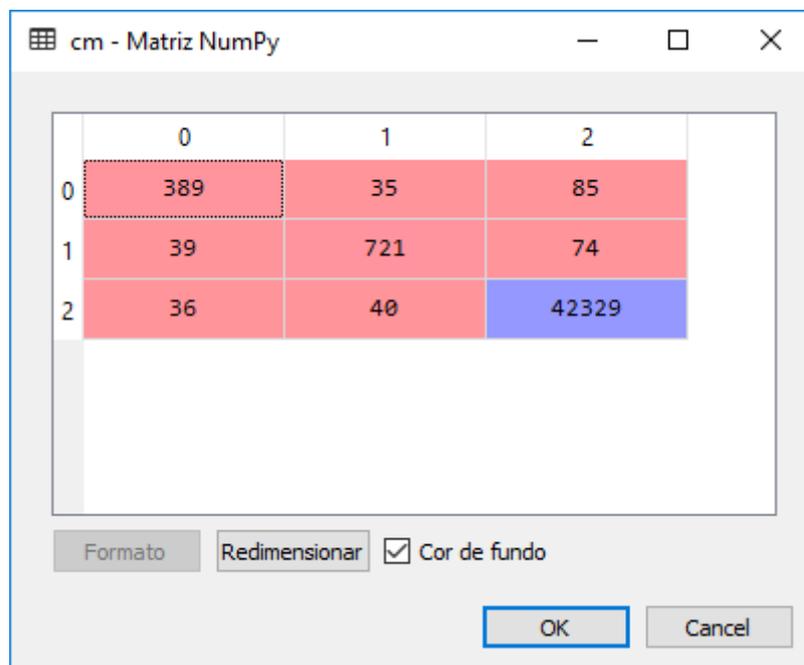


Figura 27 - Confusion Matrix da Classificação *Random Forest*

Logo, esses são os resultados, por algoritmo:

Tabela 3 - Comparação de Resultados entre Classificações

Método de Aprendizagem	Predições Corretas	Predições Incorretas	Taxa de Acerto
Regressão logística	43.418	330	99,24%
Árvore de Decisão	43.282	466	98,93%
<i>Random Forest</i>	43.439	309	99,29%

Random forest teve o melhor desempenho por precisão, seguido pela regressão logística e, por último, a árvore de decisão. Entretanto, veremos na seção 4.2.2 uma métrica mais confiável para o cálculo da precisão dos modelos. Devemos restringir o uso da *confusion matrix* para uma análise superficial.

4.1.2 Árvore de Decisão

Na Figura 28, a árvore de decisão resultante do algoritmo de árvore de decisão é apresentada. A primeira linha de cada nó informa a decisão que está sendo tomada; o caminho à esquerda representa a condição quando verdadeira, e o caminho à direita representa a condição quando falsa.

A árvore foi fixada em altura igual a 3, para facilitar a interpretação visual. Entretanto, a árvore completa possui uma altura superior a 10.

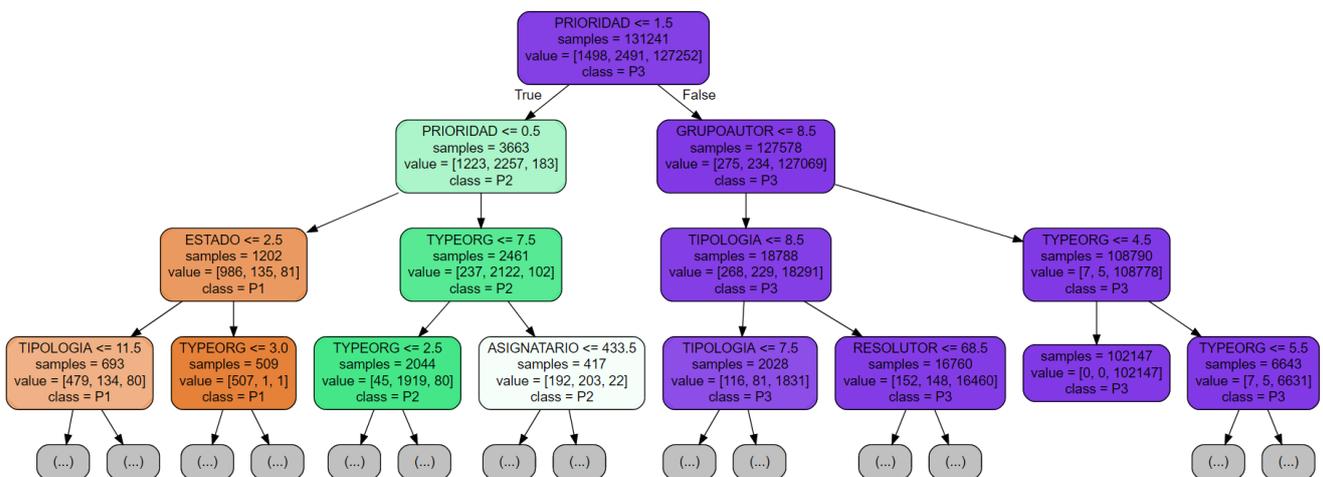


Figura 28 - Árvore de Decisão resultante da Classificação por Árvore de Decisão

4.2 Métodos de Validação e Medição de Performance

4.2.1 Accuracy Paradox

O *accuracy paradox* (em português, paradoxo de precisão), na área de análise preditiva, afirma que modelos com um nível de precisão “X” podem ter um poder de predição maior que outros modelos que possuam um nível de precisão maior [17].

Ainda que a precisão seja um atributo-chave para analisar a qualidade de um modelo de predição, ela pode provocar um erro de interpretação sobre a qualidade dessa predição. Pode ocorrer que o modelo tenha uma alta precisão, mas seja pouco útil.

Para um melhor entendimento desse paradoxo, vamos analisar dois cenários. Primeiro, a *confusion matrix* resultante do algoritmo de árvore de decisão, na Tabela 8:

Tabela 4 - Confusion Matrix da Classificação por Árvore de Decisão

Categoria/Predição	P1	P2	P3
P1	366	51	71
P2	41	732	95
P3	101	107	42.184

Como verificamos, na seção 4.1.1, a precisão dessa classificação foi de 98,93%. É um bom resultado, e vale ressaltar a complexidade computacional para que a árvore faça a classificação de cada incidente em uma categoria, como explicado na seção 2.2.1.2.

Agora, vamos considerar um cenário hipotético. Levando em conta que, em nosso cenário, a alteração de prioridade é incomum, vamos apresentar uma *confusion matrix* considerando que a nova prioridade sempre será igual a prioridade inicial do incidente. Essa simulação é apresentada na Tabela 9:

Tabela 5 - Confusion Matrix da Simulação sem Alteração de Prioridade

Categoria/Predição	P1	P2	P3
P1	342	44	26
P2	62	742	33
P3	84	82	42.333

A precisão desse “modelo”, por sua vez, será de 99,24%. Ou seja, através de uma simples regra, sem qualquer esforço preditivo ou modelo estatístico, uma precisão superior à árvore de decisão foi alcançada, pois 99,24% é maior que 98,93%. Esse fenômeno é chamado de *accuracy paradox*.

E como citado no início da sessão, isso não torna o segundo modelo melhor que o primeiro. É inútil para a empresa que se use um modelo onde o resultado da predição sempre será igual à uma das *features*. Na próxima seção, veremos um método mais apropriado para cálculo da precisão dos modelos.

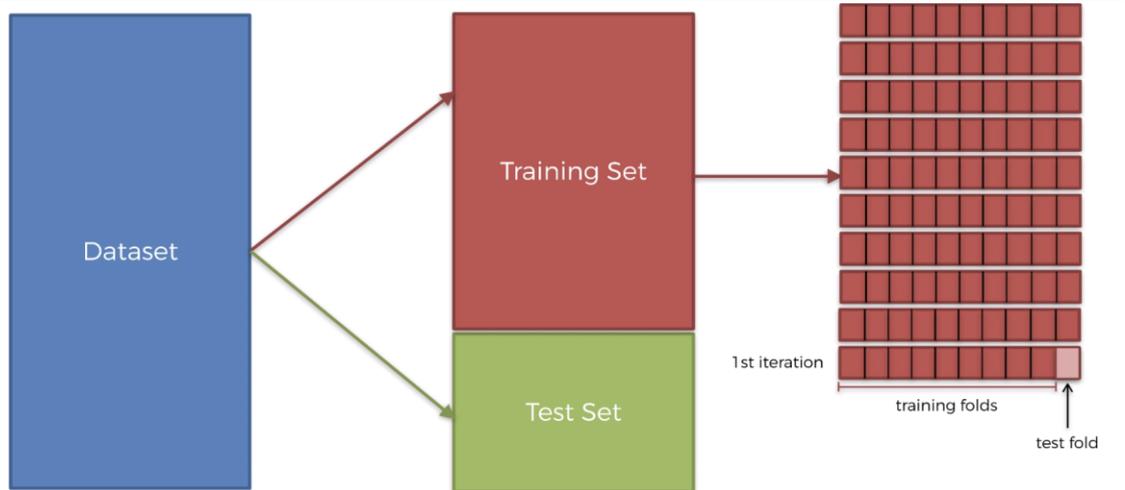
4.2.2 Validação Cruzada (Método *k-fold*)

A validação cruzada (em inglês, *k-fold cross validation*) é uma técnica muito utilizada para validação de modelos estatísticos, a fim de que seja verificada a generalização dos resultados estatísticos sobre uma base de dados independente [18].

Foi visto na seção 3.2.2.3 que os dados precisam ser divididos entre teste e treinamento onde, para o treinamento, o modelo busca identificar os padrões e, para o teste, novos dados (nunca antes vistos, pelo modelo) são testados usando os moldes obtidos pelo treinamento. O objetivo da validação cruzada é realizar uma espécie de teste, ainda na etapa de treinamento, de forma que diferentes conjuntos de dados sejam provados contra o modelo.

É chamado de “método *k-fold*” pois divide a base de treinamento em k subconjuntos, de forma que, em k interações, um dos conjuntos é usado como teste, enquanto os outros são usados para treinamento. Utilizando $k = 10$, são realizadas 10 seções de treinamento e teste, como mostra a Figura 29.

k-Fold Cross Validation



Machine Learning A-Z

© SuperDataScience

Figura 29 - Validação Cruzada 10-fold

É uma forma eficiente de:

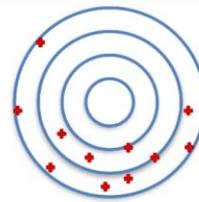
1. Reduzir *overfitting*, diversificando os dados de testes;
2. Mensurar a performance dos modelos, utilizando a média e o desvio padrão das k precisões (geralmente, $k = 10$, pois é um número suficiente para verificar a qualidade de um modelo).

Ainda sobre a segunda afirmação, a Figura 30 demonstra os cenários possíveis a respeito da avaliação um modelo. A validação cruzada, por sua natureza de testes diversificados, permite identificar essas duas fontes de erro (a tendência, em inglês, *bias*, e a variância).

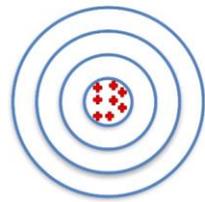
The Bias-Variance Tradeoff



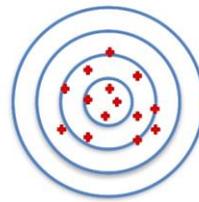
High Bias Low Variance



High Bias High Variance



Low Bias Low Variance



Low Bias High Variance

Machine Learning A-Z

© SuperDataScience

Figura 30 - Conflito Tendência x Variância

Implementando, então, a validação cruzada para os nossos algoritmos, observamos os seguintes resultados, nas Figuras 32, 34 e 36:

```
In [5]: from sklearn.model_selection import train_test_split
...: X_train, X_test, y_train, y_test = train_test_split(z, y, test_size = 0.25)

In [6]: from sklearn.linear_model import LogisticRegression
...: classifier = LogisticRegression()
...: classifier.fit(X_train, y_train)
Out[6]:
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)

In [7]: from sklearn.model_selection import cross_val_score
...: accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10, n_jobs = -1)

In [8]: accuracies.mean()
Out[8]: 0.99192323997316412

In [9]: accuracies.std()
Out[9]: 0.00068254940856976685
```

Figura 31 - Execução da Validação Cruzada para Regressão Logística

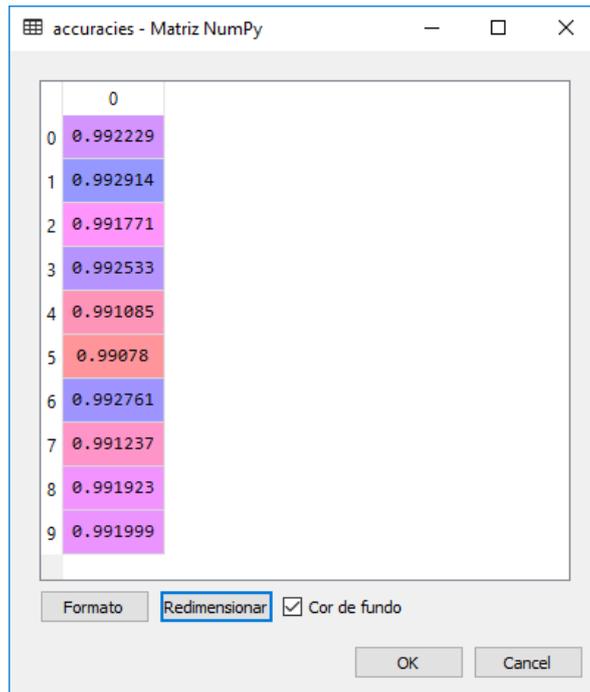


Figura 32 - Vetor de Precisões da Validação Cruzada para Regressão Logística

```
In [6]: from sklearn.model_selection import train_test_split
...: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

In [7]: from sklearn.tree import DecisionTreeClassifier
...: classifier = DecisionTreeClassifier(criterion = 'entropy')
...: classifier.fit(X_train, y_train)
Out[7]:
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=None,
splitter='best')

In [8]: from sklearn.model_selection import cross_val_score
...: accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10, n_jobs = -1)

In [9]: accuracies.mean()
Out[9]: 0.99009454832845623

In [10]: accuracies.std()
Out[10]: 0.00027085010550561688
```

Figura 33 - Execução da Validação Cruzada para Árvore de Decisão

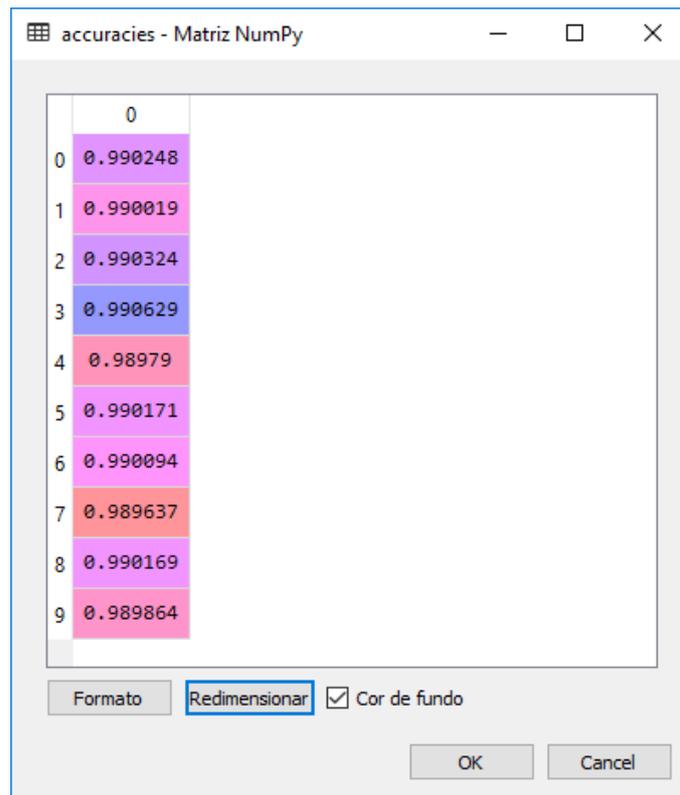


Figura 34 - Vetor de Precisões da Validação Cruzada para Árvore de Decisão

```
In [11]: from sklearn.ensemble import RandomForestClassifier
...: classifier = RandomForestClassifier(n_estimators = 100, criterion = 'entropy')
...: classifier.fit(X_train, y_train)
Out[11]:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
oob_score=False, random_state=None, verbose=0,
warm_start=False)

In [12]: from sklearn.model_selection import cross_val_score
...: accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10, n_jobs = -1)

In [13]: accuracies.mean()
Out[13]: 0.9932719292361416

In [14]: accuracies.std()
Out[14]: 0.00048540284322261826
```

Figura 35 - Execução da Validação Cruzada para *Random Forest*

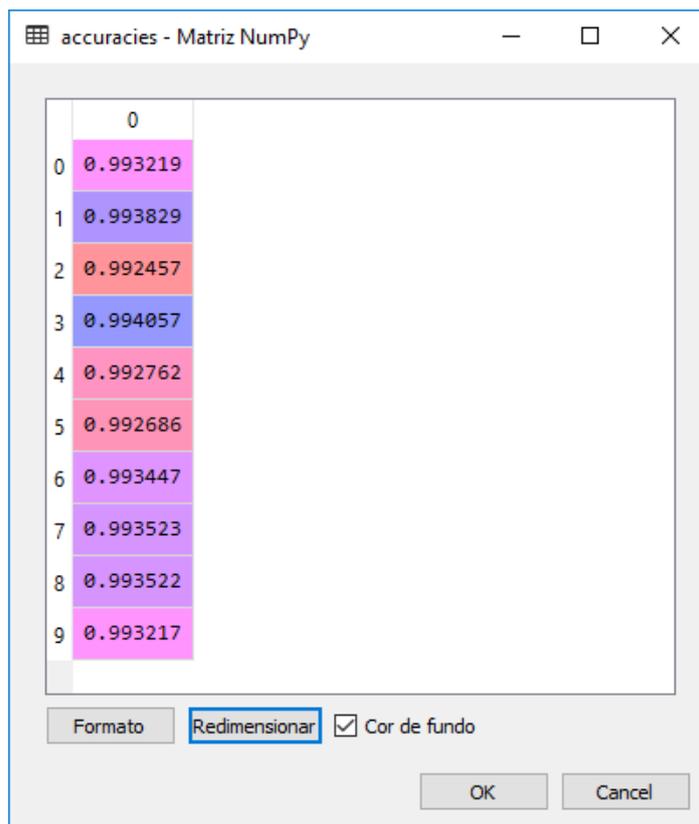


Figura 36 - Vetor de Precisões da Validação Cruzada para *Random Forest*

Em síntese, como evidenciam as últimas figuras, temos os seguintes resultados da validação cruzada:

Tabela 6 - Resultados da Validação Cruzada

Modelo	Média (10 precisões)	Desvio Padrão (10 precisões)
Regressão Logística	0,9919	0,0006
Árvore de Decisão	0,9900	0,0002
<i>Random Forest</i>	0,9932	0,0004

4.2.3 Importância de Atributos

Para que se compreenda como o modelo foi gerado, ou seja, como que se comportou após o treinamento sobre os dados, podemos utilizar um artifício para obtenção da relevância de cada atributo, em um modelo específico.

Para a regressão logística, devido a codificação *one-hot* (explicada na seção 3.2.2.1.2), existem 4985 *features*. Em outras palavras 4985 colunas ou atributos. Para obter todas as importâncias desses atributos, já que nosso modelo é uma regressão, podemos obter a lista de coeficientes de cada variável independente. Isso viabiliza a interpretação parcial do modelo, ainda que seja complexo.

A classe *LogisticRegressionClassifier* disponibiliza o atributo *coef_*, que informa a relação desses coeficientes. Esse atributo é uma matriz $A[i][j]$, onde 'i' representa cada classe do nosso problema de classificação (as prioridades: P1, P2 ou P3) e 'j' representa cada *feature*, de forma codificada.

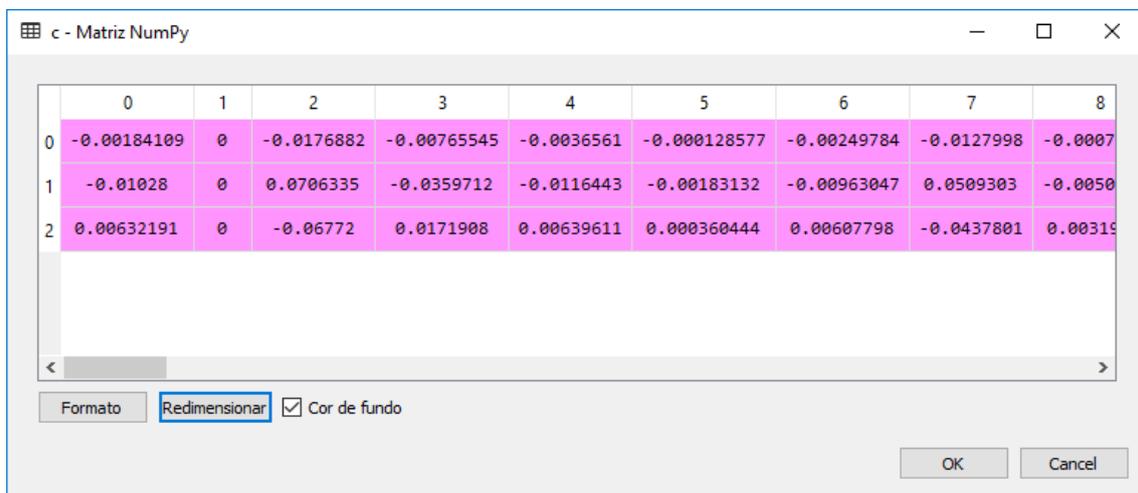


Figura 37 - Coeficientes de Variáveis Independentes na Classificação por Regressão Logística

São muitas *features*, então, Podemos explicitar apenas as 5 mais importantes para a classificação de cada classe:

Tabela 7 - Coeficientes Mais Importantes para Regressão Logística; Classe P1

Atributo	Coeficiente
3_Incidencia.Comunicaciones	3,0009
0_'G L C'	2,1825
8_MORILES	1,7854
8_RONDA-NORTE	1,7657
8_'C.T.I. SEVILLA'	1,6966

Tabela 8 - Coeficientes Mais Importantes para Regressão Logística; Classe P2

Atributo	Coeficiente
5_P2	4,3999
0_'A L E'	2,7817
0_'B S J'	2,3183
4_'CARRIZOSA MARISCAL F'	1,7993
8_'DEHESAS VIEJAS'	1,7203

Tabela 9 - Coeficientes Mais Importantes para Regressão Logística; Classe P3

Atributo	Coeficiente
5_P3	8,6015
4_'RODRIGUEZ GALAN J'	2,5271
2_'CEGES SISTEMAS'	2,2516
3_Consulta.Funcional	1,9721
4_'DIS. GRANADA SUR I'	1,9407

Para a árvore de decisão, *one-hot encoding* não foi necessário, logo, temos exatamente os 11 atributos originais da base de dados. Esses dados foram obtidos através do atributo `'classifier.feature_importances_'`, disponível na classe `DecisionTreeClassifier`.

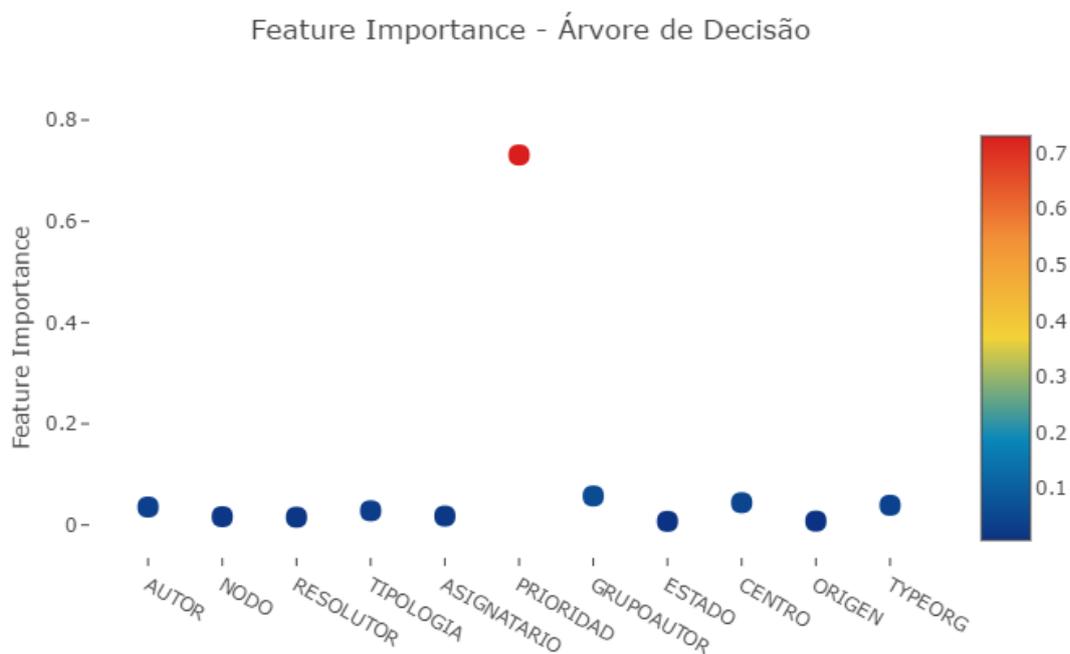


Figura 38 - Importância de *Features* para Árvore de Decisão

Tabela 10 - Lista Ordenada de Importância de Features para Árvore de Decisão

Atributo	Importância
PRIORIDAD	0,7309
GRUPOAUTOR	0,0574
CENTRO	0,044
TYPEORG	0,039
AUTOR	0,0355
TIPOLOGIA	0,0281
ASIGNATARIO	0,0177
NODO	0,0165
RESOLUTOR	0,0154
ORIGEN	0,0077
ESTADO	0,0072

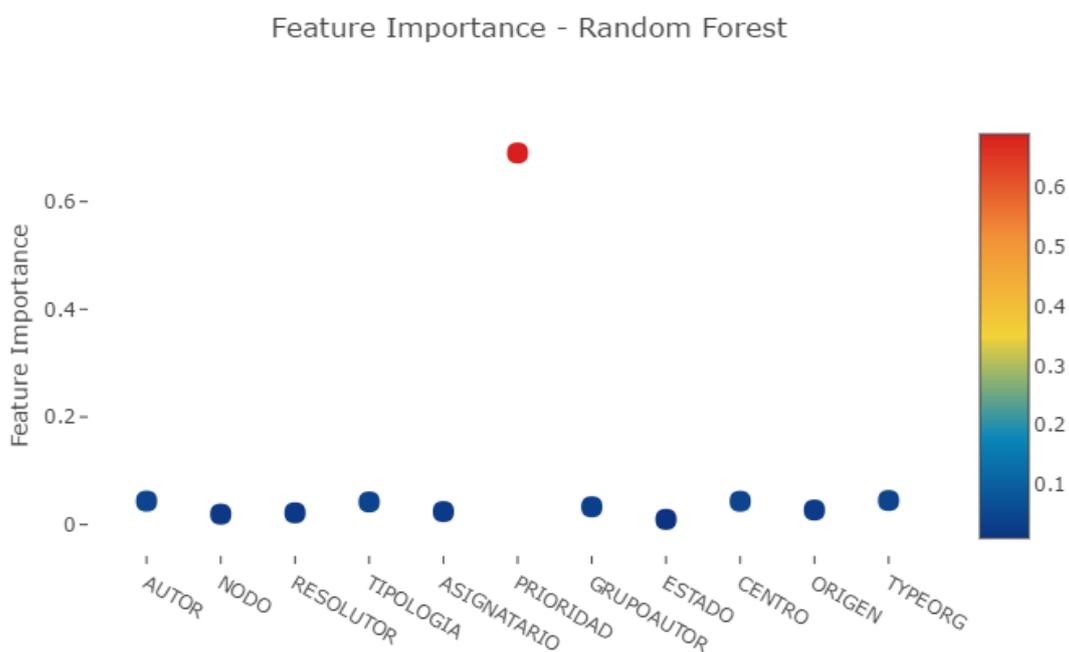


Figura 39 - Importância de Features para *Random Forest*

Tabela 11 - Lista Ordenada de Importância de Features para *Random Forest*

Atributo	Importância
PRIORIDAD	0,6902
TYPEORG	0,0447
AUTOR	0,0436
CENTRO	0,0434
TIPOLOGIA	0,0421
GRUPOAUTOR	0,0331
ORIGEN	0,0271
ASIGNATARIO	0,0242
RESOLUTOR	0,0219
NODO	0,0194
ESTADO	0,0098

5 Conclusão

5.1 Considerações Finais

Considerando todo o exposto, podemos afirmar que os três algoritmos tiveram resultados satisfatórios, com destaque para a classificação por *random forest*, que obteve o melhor resultado. Isso verifica o poder de predição dos algoritmos *ensemble* que, utilizando as capacidades de múltiplos algoritmos, simultaneamente, alcançam resultados superiores a outros algoritmos, como citado em [9].

Sobre a última análise realizada, importância de atributos, ela é um dos principais produtos do trabalho desenvolvido. A partir dos modelos, identificamos que, para o processo de negócio, atributos como o autor do incidente, o tipo de incidente e o Centro são determinantes na probabilidade de ocorrer uma mudança de prioridade; pois esses três atributos possuem alta relevância em todos os três modelos.

Em se tratando de um processo de gestão de incidentes, esses dados, de fato, deveriam ser alguns dos mais relevantes, considerando o cenário da empresa. A priorização de cada incidente é realizada por seu autor; é possível que a falha de priorização ocorra por erro humano e, uma vez que isso está identificado, através da relevância de *features*, é possível que um gestor da área tome um plano de ação sobre funcionários específicos, no âmbito de capacitar essa pessoa, reavaliar sua posição na empresa ou, por outro motivo qualquer, compreender o porque de uma alta concentração de falhas de priorização, em incidentes de um autor específico.

O tipo de incidente “Comunicações” mostrou ser o atributo mais relevante para incidentes de prioridade muito alta (P1); por muitas vezes, a natureza da demanda já é um indicador suficiente para indicar se ela é trivial ou não. Uma alta relevância desse atributo pode representar que o processo de atendimento para esse tipo de incidente não é adequado, por exemplo.

Sobre o centro responsável pelo incidente, também é um atributo-chave para a compreensão da alteração de prioridades. Nesse cenário, é possível que alguns centros tenham deficiência de atendimento sobre incidentes de uma natureza específica (informada pelos outros atributos levantados, na importância de atributos) por questões

logísticas ou de recursos humanos, por exemplo, de forma que seja necessário acionar o próximo nível de suporte.

5.2 Limitações do Projeto

As principais limitações do projeto referem-se à natureza do cenário e da empresa em questão. Uma vez que sua sede fica em Sevilla, Espanha, a comunicação foi prejudicada. Não foi feito um mapeamento de processos detalhado, de forma a compreender as especificidades do negócio.

A base de dados também possui problemas pontuais, devido a seu sistema de origem, possivelmente. O campo ‘MOTIVOCIERRE’, por exemplo, não está preenchido em diversos casos de incidentes encerrados. O campo ‘RECURSO’ não é normalizado; de forma que, aparentemente, os usuários conseguem entrar com texto livre, sem nenhuma validação; isso inviabiliza a utilização do atributo

Ainda sim, a base de dados de incidentes foi objeto suficiente para o desenvolvimento da solução de aprendizado de máquina em Python, com implementação em três algoritmos e a apresentação de resultados relevantes sobre o processo de negócio em questão.

5.3 Trabalhos Futuros

Sobre a implementação, é interessante verificar a performance de outros classificadores para esse problema, como K-NN, por exemplo; ou ainda testar diferentes parâmetros para os classificadores aqui apresentados. Verificar, também, a aplicabilidade da codificação binária em vez das codificações *one-hot* e numérica. Além disso, seria interessante aplicar o cenário do problema em redes neurais, tratando o problema como de natureza *deep learning* (aprendizado de máquina profundo).

Não se sabe sobre a performance dessa solução, no caso de redução de dimensões. É válido estudar o impacto que causaria na relação de precisão e desempenho dos algoritmos.

Por último, é adequado que o modelo construído nesse trabalho seja integrado às rotinas de monitoramento e gestão da empresa; de forma que seja possível disparar alertas por e-mail ou notificar os gestores de processo, com a maior antecedência possível, sobre previsões para incidentes de alta prioridade.

Referências Bibliográficas

- [1] IDC's Digital Universe study, sponsored by EMC (2014). Disponível em: <https://brazil.emc.com/infographics/digital-universe-2014.htm>. Acesso em Novembro de 2017.
- [2] Hegazy, Osman & Soliman, Omar S. & Abdul Salam, Mustafa. A Machine Learning Model for Stock Market Prediction. *International Journal of Computer Science and Telecommunications*. 4. 17-23, 2013.
- [3] Definição: machine learning. Enciclopedia Britannica. Disponível em: <https://global.britannica.com/technology/machine-learning>. Acesso em Novembro de 2017.
- [4] J. Shotton, "Real-Time Human Pose Recognition in Parts from a Single Depth Image", *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 1297-1304, 2011.
- [5] Baier, Thomas, Claudio Di Ciccio, Jan Mendling, and Mathias Weske. Matching events and activities by integrating behavioral aspects and label analysis. In: *Software & Systems Modeling*, pp. 1–26, 2017.
- [6] Mohri, Mehryar; Rostamizadeh, Afshin; Talwalkar, Ameet. *Foundations of Machine Learning*. USA, Massachusetts: MIT Press, 2012.
- [7] Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. *Classification and regression trees*. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software, 1984.
- [8] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*. 7 (2): 179–188, 1936.
- [9] Rokach, L. "Ensemble-based classifiers". *Artificial Intelligence Review*. 33 (1-2): 1–39, 2010.
- [10] Fernando Pérez, Brian E. Granger, IPython: A System for Interactive Scientific Computing, *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21-29, 2007.
- [11] Scikit-learn: Machine Learning in Python, Pedregosa et al., *JMLR* 12, pp. 2825-2830, 2011.
- [12] Wes McKinney. Data Structures for Statistical Computing in Python, *Proceedings of the 9th Python in Science Conference*, 51-56, 2010.

- [13] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, *Computing in Science & Engineering*, 13, 22-30, 2011.
- [14] Cawley and Talbot. On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation, *JMLR*, vol. 11, pp. 2079–2107, 2010
- [15] Kullback, S.; Leibler, R.A. On information and sufficiency. *Annals of Mathematical Statistics*. 22 (1): 79–86, 1951.
- [16] Stehman, Stephen V. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment*. 62 (1): 77–89, 1997.
- [17] Abma, B.J.M. Evaluation of requirements management tools with support for traceability-based change impact analysis. Master's thesis. The Netherlands: University of Twente. pp. 86–87, 2009.
- [18] Kohavi, Ron. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann, v. 14, p. 1137–1145, 1995.