Predição de polaridade negativa em relatórios de auditoria utilizando dados socioeconômicos

Lucas Peinado Bruscato

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Estatística

Orientador: Profa. Dra. Florencia Leonardi

São Paulo, fevereiro de 2020

Predição de polaridade negativa em relatórios de auditoria utilizando dados socioeconômicos

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 18/02/2020. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Profª. Drª. Florencia Leonardi (orientadora) - IME-USP
- Profª. Drª. Nancy Lopes Garcia - IMECC-UNICAMP
- Prof. Dr. Marcos Yamada Nakaguma - FGV

Agradecimentos

À minha mãe Helena Peinado Viotto e ao meu pai Tadeu Henrique Bruscato que não mediram esforços, em toda a vida, para que eu pudesse chegar onde cheguei, ser o que sou. Minha mãe pela persistência e meu pai pela inspiração à curiosidade. São e sempre serão meus exemplos a serem seguidos na vida.

À minha esposa e melhor amiga Natália de Deus Vilela pelo companheirismo e compreensão nesse último passo do mestrado que foi intenso e extenso, ainda mais sendo realizado em nosso novo país de residência.

À minha orientadora Florencia Leonardi pela paciência, orientação e todo conhecimento transferido nestes anos de trabalho. O apoio em realizar um trabalho com um tema considerado incomum no programa foi de uma grandeza imensurável.

Aos amigos Rachel Ubriaco por realizar críticas construtivas sobre o trabalho, Diego Wesllen por me motivar desde o primeiro momento a realizar a disciplina de Probabilidade e Inferência I e Bruno Carvalho pelas discussões técnicas que me motivaram na realização deste trabalho.

Ao Instituto de Matemática e Estatística da Universidade de São Paulo e todos os seus professores, em especial os do departamento de estatística, por fazerem parte da minha formação em estatística.

Resumo

A corrupção no Brasil afeta diretamente o bem-estar dos cidadãos ao diminuir os investimentos públicos na saúde, na educação, em infraestrutura, segurança, habitação, entre outros direitos essenciais à vida. A democratização da internet e a evolução da ciência de dados nos permitiu avaliar a relação de irregularidades administrativas, no caso deste trabalho palavras negativas, e mudanças em indicadores sociais sobre municípios. Desenvolvemos um algoritmo (web scraper) que automatiza a captura dos relatórios da auditoria da CGU e analisamos a polaridade das palavras presentes nos relatórios separadamente. Obtivemos os dados socioeconômicos no censo do IBGE em dois períodos e criamos modelos de aprendizado de máquina para predição do percentual de polaridade negativa por município baseado nos dados do IBGE. Para se avaliar a qualidade de um modelo complexo é importante ter um modelo simples como parâmetro de desempenho base, realizamos o treinamento de três modelos (regressão linear, random forest e xgboost) sobre a base de dados criada. As principais contribuições deste trabalho foram a extração automatizada dos dados governamentais, encontrar evidência estatística da relação entre os dados dos relatórios e dos dados socioeconômicos de fontes distintas e modelos de aprendizado de máquina funcionais para o problema proposto.

Palavras-chave: processamento de linguagem natural, regressão linear, random forest, xgboost, web scraper, dados socioeconômicos.

Abstract

Corruption in Brazil directly affects the well-being of citizens by diminishing public investment in health, education, infrastructure, security, housing, among other essential rights to life. The democratization of the internet and the evolution of data science allowed us to evaluate the relationship of administrative irregularities, in this case negative words, and changes in social indicators about municipalities. We developed a web scraper algorithm that automates the capture of CGU audit reports and analyzed the polarity of the words present in the reports separately. We obtained socioeconomic data from the IBGE census in two periods and created machine learning models to predict the negative polarity percentage by municipality based on IBGE data. To evaluate the quality of a complex model, it is important to have a simple model as a base performance parameter. We trained three models (linear regression, random forest and xgboost) on the created database. The main contributions of this work were the automated extraction of government data, finding statistical evidence of the relationship between reporting data and socioeconomic data from different sources and functional machine learning models for the proposed problem.

Keywords: natural language processing, linear regression, random forest, xgboost, web scraper, socioeconomic data.

Conteúdo

Capítulo 1

Introdução

1.1 Motivação

O surgimento da corrupção está relacionado à criação das primeiras civilizações e no Brasil é um problema antigo, complexo e com origem muitas vezes atribuída aos primórdios da colonização brasileira pela coroa portuguesa (ver Leite e Macedo (2017)). Durante o processo de colonização foram observados comportamentos presentes no relacionamento entre colonos e nativos que, posteriormente analisados por meio de correspondências históricas, foram classificados como corrupção conforme relatado em Leite e Macedo (2017).

É de conhecimento internacional a magnitude da corrupção no Brasil assim como suas implicações, dentre elas, pode ser citada a desigualdade social (ver Alves (2018)), que por sua vez implica em criminalidade demonstrado em Resende e Andrade (2011) e, assim, sucessivamente com uma extensa lista de causas e consequências.

Com a democratização da internet e a evolução da ciência de dados de um modo geral, diversos estudos estatísticos envolvendo transparência política e estudos relacionados com causa e consequência da corrupção foram possibilitados (e.g. Ferraz e Finan (2008) e Ransom (2013)). Tais estudos auxiliarão na identificação de corrupção e possivelmente em seu combate.

Em agosto de 2015 a Controladoria-Geral da União (CGU) iniciou um trabalho que engloba o "Programa de Fiscalização em Entes Federativos"(CGU (2015)), um novo método de controle que está sendo aplicado desde então na avaliação dos recursos públicos federais repassados a estados, municípios e Distrito Federal. No "Programa de Fiscalização em Entes Federativos"houveram fiscalizações que tiveram como objetivo avaliar a aplicacao dos recursos federais repassados aos municipios pelos Ministerios da Educacao, Saude e Integração Nacional, por exemplo. A seleção dos municípios auditados foi feita por meio de diversas normas definidas pelo CGU sendo que anualmente foram selecionados, em média, sessenta municípios.

O resultado de tal trabalho governamental - o relatório de auditoria - contém informações e conclusões dos auditores sobre cada município baseados nas informações coletadas, cuja forma segue estrita observância às normas de fiscalização aplicáveis ao Serviço Público Federal (técnicas de inspeção física e registros fotográficos, análise documental, realização de entrevistas e aplicação de questionários).

Tais relatórios possuem informações que - de uma forma automatizada por meio de técnicas de aprendizado de máquina - podem ser extraídas e interpretadas como indicadores de corrupção (desvio de verba governamental), deficiência econômica ou social.

1.2 Objetivo

O presente trabalho tem como objetivo aplicar técnicas de processamento de linguagem natural para extrair a polaridade (polaridade das palavras classificadas em positiva, neutra e negativa) média das palavras presentes em relatórios de auditoria governamentais sobre municípios brasileiros. Utilizando-se de informações sociais sobre tais municípios, aplicar técnicas de aprendizado de máquina com o intuito de predizer a média percentual de polaridade negativa de palavras presentes em tais relatórios.

1.3 Contribuições

A principal contribuição deste trabalho é propor um modelo estatístico utilizando informações dos censos do IBGE e do INEP para predição de informações extraídas de relatórios de auditoria.

Para tanto, foi feita a extração das seguintes informações: os relatórios de auditoria do "Programa de Fiscalização em Entes Federativos"(utilizados no trabalho de Ferraz e Finan (2008)), as informações do censo do IBGE (utilizadas parcialmente no trabalho de Goldani (2001)) e as informações do desempenho no ENEM (INEP). Posteriormente, foram realizadas interpretações destes relatórios utilizando aprendizado de máquina por meio de n-gram e frequência dos termos.

Foi apresentado nesta dissertação uma adição ao trabalho de Ferraz e Finan (2008) exibindo interpretações automatizadas sobre os relatórios que também foram utilizados pelos mesmos, de tal forma que a informação extraída pode ser replicada e utilizada por trabalhos futuros.

1.4 Estrutura do Trabalho

Este trabalho é organizado em 4 capítulos e um apêndice, descritos abaixo.

O primeiro capítulo - no qual o leitor se encontra - apresenta toda a estrutura, embasamento e motivação para o trabalho.

No segundo capítulo são abordados os aspectos tecnológicos utilizados no trabalho, assim como os padrões de programação escolhidos. A forma como foram coletadas e extraídas as informações de todos os relatórios produzidos pela CGU no "Programa de Fiscalização de Entes Federativos". E, por fim, a forma como foram coletadas e criadas as bases de dados com informações coletadas dos censos do IBGE e no INEP realizados nos anos 2000 e 2010.

No terceiro capítulo são analisados os dados coletados no segundo capítulo, apresentados os métodos de avaliação de desempenho, assim como o treinamento de três modelos distintos (regressão linear, random forest e xgboost) para predição do mesmo.

No quarto capítulo são apresentadas as conclusões e perspectivas para trabalhos futuros.

No apêndice são apresentados os programas para obtenção dos dados, processamento dos dados, modelagem, e tabelas com informações dos dados deste trabalho.

Capítulo 2

Contexto

Este trabalho se insere no contexto de coleta de dados estruturados e não estruturados, processamento de linguagem natural (NLP, do inglês "Natural Language Processing"), previsão de polaridade e utilização de informações socioeconômicas.

Serão utilizados os relatórios de auditoria produzidos pelo "Programa de Fiscalização de Entes Federativos"que correspondem ao período de 2011 à 2018, as informações do censo do IBGE de 2000 e 2010, as informações do ENEM (INEP) de 2000 e 2010, abordagens de NLP e aprendizado de máquina aplicadas às informações supracitadas.

Tendo como intuito garantir a total replicabilidade do trabalho, as fontes de informações foram, quase em sua completude, coletadas da internet por meio de códigos desenvolvidos neste trabalho, tornando o processo altamente automatizado. O trabalho foi construído utilizando o sistema de controle de versionamento distribuído GitHub e está disponível publicamente em Bruscato (2018), que contém os códigos criados e as informações utilizadas.

Os códigos construídos durante o desenvolvimento deste trabalho utilizaram a linguagem de programação Python (Rossum (1990)), tendo como ferramenta o Notebook Jupyter (Pérez (2015)) e norma-padrão em Python o PEP8 (Rossum (2001)) para uma melhor leitura e compreensão dos mesmos. Os códigos desenvolvidos neste trabalho foram escritos em inglês, tanto nomenclatura de variáveis quanto comentários no código.

Foi utilizada a metodologia CRISP-DM (ESPRIT (1996)) para entendimento, exploração, modelagem e avaliação do modelo na parte do trabalho que trata de ciência de dados, assim como sua proposta para estrutura de diretórios e nomenclatura de arquivos.

2.1 Programa de Fiscalização em Entes Federativos

Em agosto de 2015 a Controladoria-Geral da União (CGU) iniciou um trabalho que engloba o "Programa de Fiscalização em Entes Federativos", um novo método de controle que está sendo aplicado desde então na avaliação dos recursos públicos federais repassados a estados, municípios e Distrito Federal. Essa iniciativa, que visa inibir a corrupção entre gestores de qualquer esfera da administração pública, vem sendo aplicada desde abril de 2003 e, por meio do então "Programa de Fiscalização por Sorteios Públicos", a CGU utilizava o mesmo sistema das loterias da Caixa Econômica Federal para definir, de forma isenta, as áreas a serem fiscalizadas quanto ao correto uso dos recursos públicos federais.

Nas fiscalizações, os auditores da CGU examinam contas e documentos, além de realizarem inspeção pessoal e física das obras e serviços em andamento. Durante os trabalhos, o contato com a população, diretamente ou por meio dos conselhos comunitários e ou-

tras entidades organizadas, estimula os cidadãos a participarem do controle dos recursos oriundos dos tributos que lhes são cobrados.

O programa agora possui três formas de seleção de entes: censo, matriz de vulnerabilidade e sorteios. Nesse contexto, já foram fiscalizados cerca de 2,5 mil municípios brasileiros desde 2003, englobando recursos públicos federais superiores ao montante de R$ 30 bilhões.

Quando é utilizado o censo, a fiscalização verifica a regularidade da aplicação dos recursos em todos os entes da amostragem. Já a matriz agrega inteligência da informação, por meio da análise de indicadores, para identificar vulnerabilidades (situações locais críticas) e selecionar de forma analítica os entes a serem fiscalizados em determinada região. A metodologia de sorteios permanece aleatória, ao incorporar as ações do antigo "Programa de Fiscalização por Sorteios Públicos".

No âmbito deste trabalho serão utilizados os relatórios de auditoria realizadas no período de agosto de 2011 até junho de 2018, os quais dizem respeito aos sorteios realizados dos números 34 ao 40, que correspondem ao "Programa de Fiscalização por Sorteios Públicos", e os ciclos 3, 4 e o 5, que dizem respeito ao "Programa de Fiscalização em Entes Federativos", correspondente à 598 relatórios. Os relatórios gerados pelo programa descrito nesta seção são utilizados na criação da variável resposta do problema proposto deste trabalho. Tais relatórios já foram utilizados anteriormente em Ferraz e Finan (2008), contudo, em tal trabalho foi feita uma interpretação manual dos relatórios, i.e., um a um os relatórios foram interpretados por humanos e classificados caso tenha sido encontrado sinais de corrupção ou não no município em questão.

A coleta de tais relatórios foi realizada de forma automatizada (para as edições com respeito aos sorteios 34 ao 40) e manual (para as edições com respeito ao ciclo 3, ciclo 4 e ciclo 5). As decisões sobre as formas como os relatórios foram coletados teve embasamento na estrutura da página web do governo.

As edições com respeito aos sorteios 34 ao 40 estão disponíveis no site da Controladoria-Geral da União e podem ser encontrados por meio de um mecanismo de busca de relatórios. Contudo, a página do mecanismo de busca frequentemente se encontra indisponível e a consulta em si é onerosa de um ponto de vista de trabalho humano, dado que devem ser preenchidos diversos campos para se realizar a pesquisa. Portanto, foi desenvolvido um robô utilizando a biblioteca Selenium (Huggins (2004)) da linguagem de programação Python que auxilia na manipulação de páginas web, no caso, possibilita que de uma forma automática - utilizando códigos em Python - inicie uma instância do navegador Firefox, acesse o mecanismo de busca de relatórios da Controladoria-Geral da União e realize a consulta, um a um, dos relatórios de todos os municípios em questão.

O código citado no parágrafo anterior foi escrito na linguagem de programação Python utilizando a ferramenta Notebook Jupyter e se encontra no anexo (A).

As edições com respeito aos ciclos 3, 4 e 5 foram coletadas diretamente na página web por estarem disponíveis na página do programa mais facilmente.

2.2 SentiLex

A base de dados SentiLex-PT02 é considerada, ao menos no idioma português na atualidade, a mais importante fonte de informação no aspecto léxico de sentimento (ver Becker e Tumitan (2014)). Especificamente concebido para a análise de sentimento e opinião sobre entidades humanas em textos redigidos em português, é constituído atualmente por 7.014 lemas e 82.347 formas flexionadas.

Os adjetivos presentes na base possuem uma polaridade (polaridade de cada palavra, sendo positiva, neutra ou negativa) que foi atribuída com base num cálculo sobre as distâncias das palavras, com polaridade conhecida a priori, ligadas aos adjetivos por uma relação de sinonímia num grafo, inferido a partir de dicionários de sinônimos disponíveis para o português. Os detalhes da criação de tal trabalho podem ser encontrados em Silva e Carvalho (2012).

Cada palavra presente na base de dados possui sua respectiva polaridade, sendo que os valores que as polaridades podem assumir são -1, 0 e 1, representando polaridade negativa, polaridade neutra e polaridade positiva, respectivamente.

2.3 Variável Resposta

Para realizar uma interpretação quantitativa sobre os relatórios coletados que foram apresentados na Seção "Programa de Fiscalização de Entes Federativos", este trabalho utiliza a base da dados SentiLex para definição de polaridade de cada uma das palavras presentes nos relatórios e, assim, criação de uma métrica para modelagem.

Foi criado um código que realiza o reconhecimento de caracteres do documento pdf onde, iterativamente utilizando regras para separadores de palavras, se concatenou caracter a caracter formando as palavras do relatório pdf. Devido a problemas enfrentados com respeito a diferentes tipos de codificação (encoding, e.g., UTF-8) dos relatórios utilizados, foi necessário a utilização da biblioteca externa em python PDF Miner (Shinyama (2004)) para extração das palavras. Assim, a construção de tal métrica se deu pela criação de um código em python que realiza o reconhecimento das palavras presentes em documentos pdf. Pode-se sumarizar as etapas do processo pelos seguintes passos:

1. Iterativamente execute o código da biblioteca externa (PDF Miner) para cada relatório de auditoria listado na Seção "Programas de Entes Federativos"(2.1)

2. Sobre o texto de cada relatório remova acentuação e transforme todos os caracteres em minúsculo

3. Calcule a frequência das palavras dentro de seu respectivo documento

4. Utilizando a base de dados SentiLex (extraída por meio da criação de um código em python), apresentada na Seção "SentiLex"(2.2), defina a polaridade da palavra em positiva, negativa ou neutra

5. Realize um agrupamento sobre o produto dos passos anteriores sumarizando a frequência relativa calculada sob as polaridades definidas, i.e., o resultado é um arquivo em formato CSV com as seguintes colunas: nome do relatório, quantidade de palavras, percentual de polaridade negativa, percentual de polaridade positiva, percentual de polaridade neutra e percentual de polaridade (ou, nesse sentido, palavra) não encontrada

Para o item 5 do processo acima, foram calculados a mediana, o mínimo, o máximo e o desvio-padrão dos percentuais de polaridade e estão apresentados na tabela abaixo.

Tabela 2.1: *Métricas de percentual de polaridade*

Métrica	Negativa	Positiva	Neutra	Não Encontrada
mediana	1.70%	2.79%	0.74%	94.91%
mínimo	0.36%	1.49%	0.18%	92.76%
máximo	4.54%	4.54%	1.79%	97.04%
desvio-padrão	0.37%	0.35%	0.18%	0.59%

Os códigos citados no parágrafo anterior foram escritos na linguagem de programação Python utilizando a ferramenta Notebook Jupyter e se encontram nos anexos A.4 e A.5, respectivamente.

O resultado da execução dos itens enumerados acima proporciona a criação da variável resposta, que é dada como sendo o percentual de palavras negativas encontradas em um relatório dividido pela soma do valor do percentual de palavras negativas e do valor do percentual de palavras positivas, ou seja, se relativiza o percentual de palavras negativas pela soma do percentual de palavras negativas e percentual de palavras positivas para, assim, facilitar a interpretação dos resultados e desconsiderar o percentual de palavras neutras e o percentual de palavras não encontradas.

O agrupamento realizado no item 5 do processo descrito para extração das métricas se assemelha à medida estatística tf-idf (term frequency–inverse document frequency, ver Spärck Jones (1972)), que tem o intuito de indicar a importância de uma palavra de um documento em relação a uma coleção de documentos ou em um corpus linguístico. O valor tf–idf de uma palavra aumenta proporcionalmente à medida que aumenta o número de ocorrências dela em um documento, no entanto, esse valor é equilibrado pela frequência da palavra no corpus. Isso auxilia a distinguir o fato da ocorrência de algumas palavras serem geralmente mais comuns que outras. Contudo, devido aos relatórios terem sido coletados em diferentes anos por diferentes auditores que utilizaram-se de vocabulários e termos distintos, tal técnica não foi utilizada e a métrica calculada foi apenas a frequência dentro do próprio documento e não ponderado pelo conjunto de documentos tal como a medida estatística tf-idf.

2.4 Censo do IBGE

O censo demográfico no Brasil é uma operação censitária realizada a nível nacional a partir do ano de 1872, constitui a principal fonte de referência para o conhecimento das condições de vida da população em todos os municípios do País e em seus recortes territoriais internos, tendo como unidade de coleta a pessoa residente, na data de referência, em domicílio do Território Nacional.

O Instituto Brasileiro de Geografia e Estatística (IBGE) é o órgão responsável por realizar o censo demográfico brasileiro a partir do ano de 1940, sendo o último censo tendo sido realizado no ano de 2010 e o próximo previsto para acontecer em 2020.

O Questionário Básico da pesquisa investiga informações sobre características dos domicílios (condição de ocupação, número de banheiros, existência de sanitário, escoadouro do banheiro ou do sanitário, abastecimento de água, destino do lixo, existência de energia elétrica etc.); emigração internacional; composição dos domicílios (número de moradores, responsabilidade compartilhada, lista de moradores, identificação do responsável, relação de parentesco com o responsável pelo domicílio etc.); características do morador (sexo e idade, cor ou raça, etnia e língua falada, no caso dos indígenas, posse de registro de

nascimento, alfabetização, rendimento mensal etc.); e mortalidade. A investigação nos domicílios selecionados, efetuada por meio do Questionário da Amostra, inclui, além dos quesitos presentes no Questionário Básico, outros mais detalhados sobre características do domicílio e das pessoas moradoras, bem como quesitos sobre temas específicos, como deficiência, nupcialidade e fecundidade.

A periodicidade da pesquisa é decenal, excetuando-se os anos de 1910 e 1930, em que o levantamento foi suspenso, e 1990, quando a operação foi adiada para 1991. Sua abrangência geográfica é nacional, com resultados divulgados para Brasil, Grandes Regiões, Unidades da Federação, Mesorregiões, Microrregiões, Regiões Metropolitanas, Municípios, Distritos, Subdistritos e Setores Censitários.

As informações utilizadas neste trabalho são as disponibilizadas pelos censos realizados nos anos 2000 e nos anos 2010 sob a perspectiva de comparação das informações de ambos censos. Devido à mudança na informação ou segmentação realizada em cada censo, foi realizado um mapeamento de todas as informações disponíveis por censo para definição de quais informações seriam utilizadas no trabalho (a estatística levantada no censo de 2000 pode não existir no censo de 2010, e vice-versa, ou uma segmentação diferente foi realizada em cada censo impossibilitando a comparação da informação entre ambos). Os dados coletados do IBGE são números absolutos, portanto, tais números foram divididos pelo número de habitantes do município em questão obtendo-se assim o percentual relativo pelo ano em questão. As informações que serão utilizadas estão descritas abaixo.

- `education_var_01_quantity_pct`: Percentual de pessoas que frequentavam creche ou escola por município em 2000 dividido pela mesma informação em 2010
- `family_var_01_suitable_pct`: Percentual de domicílios particulares permanentes em situação adequada por município em 2000 dividido pela mesma informação em 2010
- `family_var_01_semi_suitable_pct`: Percentual de domicílios particulares permanentes em situação semi-adequada por município em 2000 dividido pela mesma informação em 2010
- `family_var_01_inappropriate_pct`: Percentual de domicílios particulares permanentes em situação inadequada por município em 2000 dividido pela mesma informação em 2010
- `fertility_var_01_has_children_pct`: Percentual de mulheres de 10 anos ou mais de idade total no período de referência de 12 meses anteriores ao Censo por município em 2000 dividido pela mesma informação em 2010
- `fertility_var_01_children_born_pct`: Percentual de mulheres de 10 anos ou mais de idade que tiveram filhos nascidos no período de referência de 12 meses anteriores ao Censo por município em 2000 dividido pela mesma informação em 2010
- `fertility_var_01_children_borned_live_pct`: Percentual de mulheres de 10 anos ou mais de idade que tiveram filhos nascidos vivos no período de referência de 12 meses anteriores ao Censo por município em 2000 dividido pela mesma informação em 2010

- `fertility_var_01_children_borned_dead_pct`: Percentual de mulheres de 10 anos ou mais de idade que tiveram filhos nascidos mortos no período de referência de 12 meses anteriores ao Censo por município em 2000 dividido pela mesma informação em 2010
- `fertility_var_02_married_pct`: Percentual de pessoas de 10 anos ou mais de idade casadas por município em 2000 dividido pela mesma informação em 2010
- `fertility_var_02_separated_pct`: Percentual de pessoas de 10 anos ou mais de idade separadas por município em 2000 dividido pela mesma informação em 2010
- `fertility_var_02_divorced_pct`: Percentual de pessoas de 10 anos ou mais de idade divorciadas por município em 2000 dividido pela mesma informação em 2010
- `fertility_var_02_widow_pct`: Percentual de pessoas de 10 anos ou mais de idade viúvas por município em 2000 dividido pela mesma informação em 2010
- `fertility_var_02_single_pct`: Percentual de pessoas de 10 anos ou mais de idade solteiras por município em 2000 dividido pela mesma informação em 2010
- `fertility_var_03_total_pct`: Percentual de pessoas de 10 anos ou mais de idade que viviam em companhia de cônjuge ou companheiro(a), por natureza da união, por município em 2000 dividido pela mesma informação em 2010
- `work_var_01_regular_pct`: Percentual de pessoas de 10 anos ou mais de idade empregadas em situação regular por município em 2000 dividido pela mesma informação em 2010
- `work_var_01_irregular_pct`: Percentual de pessoas de 10 anos ou mais de idade empregadas em situação irregular por município em 2000 dividido pela mesma informação em 2010
- `social_indicator_var_01_15_to_24_years_pct`: Taxa de analfabetismo da população de 15 a 24 anos por município em 2000 dividido pela mesma informação em 2010
- `social_indicator_var_01_25_to_59_years_pct`: Taxa de analfabetismo da população de 25 a 59 anos por município em 2000 dividido pela mesma informação em 2010
- `social_indicator_var_01_60_to_more_years_pct`: Taxa de analfabetismo da população de 60 ou mais anos por município em 2000 dividido pela mesma informação em 2010
- `social_indicator_var_02_suitable_pct`: Proporção de domicílios particulares permanentes em situação apropriada por município em 2000 dividido pela mesma informação em 2010
- `social_indicator_var_02_semi_suitable_pct`: Proporção de domicílios particulares permanentes em situação semi-apropriada por município em 2000 dividido pela mesma informação em 2010

- `social_indicator_var_02_inappropriate_pct`: Proporção de domicílios particulares permanentes em situação inapropriada por município em 2000 dividido pela mesma informação em 2010
- `social_indicator_var_03_responsable_illiterate_pct`: Proporção de crianças de 0 a 5 anos de idade residentes em domicílios particulares permanentes com responsável ou cônjuge analfabeto por município em 2000 dividido pela mesma informação em 2010
- `social_indicator_var_03_inappropriate_residence_pct`: Proporção de crianças de 0 a 5 anos de idade residentes em domicílios particulares permanentes com saneamento inadequado por município em 2000 dividido pela mesma informação em 2010
- `social_indicator_var_03_responsable_illiterate_and_inappropriate_residence_pct`: Proporção de crianças de 0 a 5 anos de idade residentes em domicílios particulares permanentese com responsável ou cônjuge analfabeto e saneamento inadequado por município em 2000 dividido pela mesma informação em 2010

Os relatórios contendo as estatísticas do censo do IBGE de 2000 e 2010 foram coletados manualmente no site do IBGE (IBGE (2000)), contudo, foram criados dois códigos para extração estruturada da informação. O primeiro foi para organização dos diretórios dos relatórios baseado na sigla do estado, e o segundo para extração das informações das planilhas em extensão XLS (eXceL Spreadsheet) para o formato de arquivos CSV (Comma-Separated Values). A motivação para o segundo código se deve ao fato de que as planilhas originais da fonte contém formatação com layout disforme, de forma que se tornava impossível a extração direta da informação de forma simplificada.

Os códigos citados no parágrafo anterior foram escritos na linguagem de programação Python utilizando a ferramenta Notebook Jupyter e se encontram nos anexos A.1 e A.2, respectivamente.

2.5 Enem (INEP)

O Exame Nacional do Ensino Médio (Enem) é uma prova realizada pelo Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (INEP), autarquia vinculada ao Ministério da Educação do Brasil, e foi criada em 1998. Ela é utilizada para avaliar a qualidade do ensino médio no país. Seu resultado serve para acesso ao ensino superior em universidades públicas brasileiras, através do Sistema de Seleção Unificada (SiSU), assim como em algumas universidades no exterior.

A prova também é feita por pessoas com interesse em ganhar bolsa integral ou parcial em universidade particular através do Programa Universidade para Todos (ProUni) ou para obtenção de financiamento através do Fundo de Financiamento ao Estudante do Ensino Superior (Fies). Entre 2009 e 2016 o exame serviu também como certificação de conclusão do ensino médio em cursos de Educação de Jovens e Adultos (EJA), antigo supletivo, substituindo o Exame Nacional para Certificação de Competências de Jovens e Adultos (Encceja), que voltou a ser realizado a partir de 2017. Em algumas universidades, substituiu o tradicional vestibular.

A prova foi criada em 1998, sendo usada inicialmente para avaliar a qualidade da educação nacional e já na segunda edição do exame, foi utilizada como modalidade de acesso ao ensino superior. Teve sua segunda versão iniciada em 2009, com aumento do número de questões e utilização da prova em substituição ao antigo vestibular. O exame é realizado anualmente e tem duração de dois dias, contém 180 questões objetivas (divididas em quatro grandes áreas) e uma questão de redação.

As informações utilizadas neste trabalho são as disponibilizadas pela plataforma do INEP na Seção "Microdados"(IBGE (2000)), e foram selecionados os dados dos exames realizados em 2000 e em 2010 sob a perspectiva de comparação das informações de ambos anos. As estatísticas extraídas foram:

- `enem_var_01_enem_score_mean_pct`: Média da nota final do enem em 2000 dividido pela mesma informação em 2010
- `enem_var_01_enem_score_std_pct`: Desvio-padrão da nota final do enem em 2000 dividido pela mesma informação em 2010
- `enem_var_01_enem_score_median_pct`: Mediana da nota final do enem em 2000 dividido pela mesma informação em 2010

O código citado no parágrafo anterior foi escrito na linguagem de programação Python utilizando a ferramenta Notebook Jupyter e se encontra no anexo A.3.

2.6 Variáveis Explicativas

A variável que define o estado do município em questão foi utilizada como variável explicativa, contudo, foi-se necessário uma binarização da mesma utilizando um método conhecido como one-hot-encoding. O método realiza a transformação de uma variável categórica em várias variáveis numéricas, sendo que para cada categoria de tal variável é criado uma nova variável contendo o valor zero ou um, sendo que zero indica a não ocorrência de tal categoria na observação em questão e um indica a ocorrência de tal categoria na observação em questão. Tal transformação se fez necessária uma vez que os modelos de aprendizado de máquina implementados na biblioteca scikit-learn em Python não aceitam como entrada bases de dados contendo variáveis categóricas.

As variáveis extraídas do censo demográfico do IBGE dos anos 2000 e 2010 também foram utilizadas como explicativas para modelagem do problema proposto. A abordagem utilizada para tais variáveis foi utilizar as informações de modo comparativo. As variáveis do censo do IBGE que não são representadas como porcentagens foram divididas numericamente pelo tamanho populacional do município em questão, assim, se tornando porcentagens com respeito ao tamanho populacional.

Após a criação das novas variáveis conforme descrito no parágrafo anterior, todas as variáveis (tanto as variáveis criadas conforme descrição do parágrafo anterior quanto as variáveis do censo do IBGE que são representadas como porcentagens) correspondentes ao censo de 2000 foram divididas numericamente pelas variáveis de mesmo conceito correspondentes ao censo de 2010. As variáveis criadas estão na tabela abaixo e as variáveis de origem utilizadas foram descritas na Seção 2.4.

Tabela 2.2: *Nome de todas as variáveis explicativas utilizadas*

Nome das variáveis
education_var_01_quantity_pct
family_var_01_suitable_pct
family_var_01_semi_suitable_pct
family_var_01_inappropriate_pct
fertility_var_01_has_children_pct
fertility_var_01_children_born_pct
fertility_var_01_children_borned_live_pct
fertility_var_01_children_borned_dead_pct
fertility_var_02_married_pct
fertility_var_02_separated_pct
fertility_var_02_divorced_pct
fertility_var_02_widow_pct
fertility_var_02_single_pct
fertility_var_03_total_pct
work_var_01_regular_pct
work_var_01_irregular_pct
social_indicator_var_01_15_to_24_years_pct
social_indicator_var_01_25_to_59_years_pct
social_indicator_var_01_60_to_more_years_pct
social_indicator_var_02_suitable_pct
social_indicator_var_02_semi_suitable_pct
social_indicator_var_02_inappropriate_pct
social_indicator_var_03_responsable_illiterate_pct
social_indicator_var_03_inappropriate_residence_pct
social_indicator_var_03_responsable_illiterate_and_inappropriate_residence_pct
enem_var_01_enem_score_mean_pct
enem_var_01_enem_score_std_pct
enem_var_01_enem_score_median_pct
state_ac
state_al
state_ba
state_ce
state_es
state_go
state_ma
state_mg
state_ms
state_mt
state_pa
state_pb
state_pe
state_pi
state_pr
state_rj
state_rn
state_ro
state_rr
state_rs
state_sc
state_se
state_sp
state_to

As variáveis extraídas do enem do INEP nos anos 2000 e 2010 também foram utilizadas como explicativas para modelagem do problema proposto. A abordagem utilizada para tais variáveis foi utilizar as informações de modo comparativo. As variáveis correspondentes ao exame aplicado em 2000 foram divididas numericamente pelas variáveis de mesmo conceito correspondentes ao exame aplicado em 2010. As variáveis criadas estão na tabela 2.2 e as variáveis brutas foram descritas na Seção 2.5.

O código para criação das variáveis citadas nesta seção foi escrito na linguagem de programação Python utilizando a ferramenta Notebook Jupyter e se encontra no anexo A.7.

2.7 Criação da base para modelagem

Utilizando como chave estrangeira a informação de munício e estado, realizou-se - por meio do código em python no anexo A.6 - a união das informações citadas nas Seções "Variável Resposta"(2.3) e "Variáveis Explicativas"(2.6) e, assim, foram obtidas as bases para modelagem do problema proposto neste trabalho. Por meio do método de amostragem aleatória selecionou-se setenta e cinco por cento da base gerada para desenvolvimento do modelo e vinte e cinco por cento para validação do desempenho do modelo, foi-se utilizado o método de validação cruzada descrita em Kohavi (1995) (amplamente aplicado em diversos trabalhos, tais como em Wesllen (2013)).

O código desenvolvido para criação das bases de desenvolvimento e validação citadas no parágrafo anterior foi escrito na linguagem de programação Python utilizando a ferramenta Notebook Jupyter e se encontra no anexo A.8.

Capítulo 3

Modelagem

Utilizando-se da base de dados para modelagem descrita na Seção "Criação da base para modelagem"(2.7) - cujas variáveis explicativas foram descritas na Seção "Variáveis Explicativas"(2.6) e a variável resposta descrita na Seção "Variável Resposta"(2.3) - foram aplicados modelos de aprendizado de máquina com o intuito de obter o melhor desempenho para predição da variável resposta utilizando as variáveis explicativas construídas.

Os métodos de aprendizado de máquina utilizados foram Regressão Linear, Random Forest e XGBoost, sendo que a afinação dos parâmetros de tais modelos foi feita utilizando-se o método Randomized Search. A avaliação de desempenho destes modelos foi feita utilizando-se de duas métricas, a raiz do erro quadrático médio e as bandas de acerto, e a interpretação dos modelos Random Forest e XGBoost foram feitas por meio do método Shapley Value.

3.1 Análise Descritiva

A análise descritiva consiste em explorar dados utilizados para a modelagem com o intuito de encontrar a existência, ou não, de uma relação causal entre as variáveis explicativas e a variável resposta. Neste trabalho já foram apresentadas algumas análises descritivas no Capítulo 2, como, por exemplo, as estatísticas dos percentuais de polaridade obtidos da variável resposta.

Foram utilizadas 52 variáveis explicativas neste trabalho, foi realizado uma análise em cada uma das variáveis utilizando-se de gráficos de dispersão com a variável resposta. Nestes gráficos de dispersão, os valores observados para a variável explicativa estão dispostos no eixo das ordenadas e os valores observados para a variável resposta estão dispostos no eixo das abscissas. Como exemplo, o gráfico de dispersão - que se encontra no anexo B.1 - da variável `'family_var_01_inappropriate_pct'` (que se trata do percentual de domicílios particulares permanentes em situação inadequada por município em 2000 dividido pela mesma informação em 2010), nota-se uma relação linear positiva média entre a variável explicativa e a variável resposta. Os gráficos de dispersão de todas as variáveis explicativas se encontram no anexo B.1.

O código citado no parágrafo anterior foi escrito na linguagem de programação Python utilizando a ferramenta Notebook Jupyter e se encontra no anexo A.9.

3.2 Randomized Search

O Randomized Search é um método utilizado para otimização de hiperparâmetros que consiste em uma pesquisa aleatória sobre os hiperparâmetros fornecidos para treinamento de modelos de aprendizado de máquina visando o melhor desempenho de tal modelo. Neste trabalho foi escolhido utilizar-se o Randomized Search frente aos dois métodos mais utilizados na atualidade - o Grid Search (uma pesquisa exaustiva por meio de um subconjunto especificado manualmente do espaço de hiperparâmetros de um algoritmo de aprendizado) e a busca manual - uma vez que o Randomized Search realiza buscas mais eficientes que o Grid Search e a busca manual (ver Bergstra e Bengio (2012)).

Foi-se utilizada a implementação deste método proveniente da biblioteca em Python Scikit-learn (Cournapeau (2007)). Ao contrário do Grid Search, nem todos os valores dos parâmetros são testados, mas um número fixo de configurações de parâmetros é amostrado das distribuições especificadas. O número de configurações de parâmetros que são tentadas é definido pela quantidade de iterações passadas ao método e o desempenho do modelo de aprendizado de máquina treinado após a escolha dos hiperparâmetros é calculado utilizando-se do método de validação cruzada.

Se todos os parâmetros forem apresentados como uma lista, é realizada a amostragem sem reposição. Se pelo menos um parâmetro for fornecido como distribuição, é utilizada a amostragem com reposição. No âmbito deste trabalho, alguns parâmetros foram fornecidos como lista e outros como distribuição, assim, para o treinamento do modelo de aprendizado de máquina Random Forest foi-se utilizado uma amostragem com reposição e para o treinamento do modelo de aprendizado de máquina XGBoost foi-se utilizado uma amostragem sem reposição.

Além do Randomized Search ser a melhor opção por sua eficiência, existe a motivação da viabilidade da otimização dos hiperparâmetros, uma vez que para o treinamento do modelo de aprendizado de máquina XGBoost foi-se utilizado dez diferentes hiperparâmetros com diferentes quantidades de valores, representando no final uma quantidade de 1.411.200 possibilidades de cenários distintos para o treinamento.

3.3 Shapley Value

O Shapley Value é uma proposta de solução na teoria dos jogos cooperativos, foi nomeado em homenagem a Lloyd Shapley que o definiu inicialmente (ver Shapley e Tucker (1953)).

Em um cenário de jogos cooperativos onde, para cada jogo cooperativo, atribui-se uma distribuição única (entre os jogadores) de um excedente total gerado pelo grupo de todos os jogadores, o Shapley Value é caracterizado como uma coleção de propriedades para explicação da proveniência de cada parte desse excedente total.

A configuração é da seguinte forma: um grupo de jogadores coopera e obtém um certo ganho geral com essa cooperação. Como alguns jogadores podem contribuir mais para o grupo do que outros, ou podem ter um poder de barganha diferente, qual a importância de cada jogador para a cooperação geral e que recompensa ele pode razoavelmente esperar? Esta pergunta é uma das motivações para criação do Shapley Value, i.e., explicar essa importância individual e a recompensa esperada.

O paralelo da explicação acima à explicação do algoritmo em aprendizado de máquina seria de que os jogadores são as variáveis explicativas, o ganho geral obtido por cada jogador é o impacto na predição do modelo e a importância individual seria a importância da

variável explicativa no modelo. A importância das variáveis explicativas em modelos de aprendizado de máquina é uma informação amplamente discutida, uma vez que existem diferentes formas de se calcular tal importância e, muitas vezes, a ordenação da importância das variáveis pode ser diferente de um método para outro gerando discussão sobre a escolha do melhor método a ser utilizado.

O Shapley Value vem sendo considerado a forma mais robusta e consistente de se avaliar a importância das variáveis explicativas em um modelo (ver Lundberg e Lee (2017)), ele será utilizado neste trabalho tanto para ordenação da importância das variáveis quanto para interpretação das variáveis explicativas utilizadas no modelo. Foi-se utilizada a implementação deste método proveniente da biblioteca em Python SHAP (Lundberg (2018)).

Uma simplificação de como o Shapley Value pode ser calculado para uma determinada variável explicativa j pode ser dada pela fórmula abaixo (considere f o modelo que se deseja avaliar o impacto das variáveis explicativas).

$$\hat{\phi}_j = \frac{1}{M}\sum_{m=1}^{M}(\hat{f}(x_{+j}^m) - \hat{f}(x_{-j}^m)) \tag{3.1}$$

Onde,

- $\hat{f}(x_{+j}^m)$ é a predição para x_{+j}^m, sendo que x_{+j}^m é um vetor de variáveis explicativas provindas de uma observação aleatória z (z é uma observação aleatória selecionada da base de treinamento), exceto pelo valor da variável explicativa que corresponde à j.
- o vetor x_{-j}^m é quase idêntico ao vetor x_{+j}^m com a diferença de que o valor x_j^m é tomado da observação aleatória z.

Assim, o algoritmo para se calcular o Shapley Value é dado por:

- Para $m = 1, ..., M$, onde M é o total de iterações e p é o total de variáveis explicativas:
  - Selecione duas observações aleatórias (vetores x e z) de todas observações disponíveis na base de treinamento
  - Selecione uma permutação aleatória das variáveis explicativas
  - Obtenha o vetor $x_0 = (x_{(1)}, ..., x_{(j)}, ..., x_{(p)})$
  - Obtenha o vetor $z_0 = (z_{(1)}, ..., z_{(j)}, ..., z_{(p)})$
  - Construa dois vetores:
    - Com a variável j: $x_{+j} = (x_{(1)}, ..., x_{(j-1)}, x_{(j)}, z_{(j+1)}, ..., z_{(p)})$
    - Sem a variável j: $x_{-j} = (x_{(1)}, ..., x_{(j-1)}, z_{(j)}, z_{(j+1)}, ..., z_{(p)})$
  - Calcule a contribuição marginal: $\phi_j^m = \hat{f}(x_{+j}^m) - \hat{f}(x_{-j}^m)$
- Calcule o Shapley Value como a média: $\hat{\phi}_j = \frac{1}{M}\sum_{m=1}^{M}(\phi_j^m)$

3.4 Métricas para avaliação de desempenho

Neste trabalho serão utilizadas duas métricas para avaliação de desempenho, a raiz do erro quadrático médio e a banda de acerto.

A raiz do erro quadrático médio (do inglês root mean squared error, RMSE) é uma medida frequentemente usada das diferenças entre os valores previstos por um modelo ou estimador e os valores observados. O RMSE representa a raiz quadrada da média quadrática das diferenças entre os valores previstos e os valores observados. Esses desvios são chamados de resíduos, quando os cálculos são realizados sobre a amostra de dados usada para estimativa, ou são chamados de erros (ou erros de previsão), quando calculados fora da amostra. O RMSE é uma medida de precisão, para comparar erros de previsão de diferentes modelos para um conjunto de dados específico e não entre conjuntos de dados, pois depende da escala.

O RMSE é sempre não negativo e um valor 0 (quase nunca alcançado na prática) indicaria um ajuste perfeito para os dados. Em geral, um RMSE mais baixo é melhor que um mais alto. No entanto, comparações entre diferentes tipos de dados seriam inválidas porque a medida depende da escala dos números usados. Neste trabalho, as medidas de RMSE são comparáveis uma vez que os dados trabalhados são os mesmos mudando apenas os modelos.

$$\mathbf{RMSE} = \sqrt{\frac{\sum_{t=1}^{N}(\hat{y}_t - y_t)^2}{N}} \tag{3.2}$$

Sendo que,

- $\hat{y}_t$ representa o valor predito do modelo
- y_t representa o valor observado
- N representa a quantidade de observações

A banda de acerto é uma medida para avaliar a distribuição dos erros de predição separando-os em três categorias interpretáveis. De acordo com o valor de predição do modelo, se classifica tal predição em cada banda, tais bandas são catalogadas como "subestimado", "bem estimado"e "superestimado". As regras para classificação em cada uma dessas opções estão listadas abaixo. O valor de δ foi determinado empiricamente após sucessivos testes como sendo 0.05.

- Subestimado: Se o valor da predição for menor que o valor observado menos δ
- Superestimado: Se o valor da predição for maior que o valor observado mais δ
- Bem estimado: Se o valor da predição estiver entre o valor observado menos δ e o valor observado mais δ

3.5 Regressão Linear

A regressão linear é uma abordagem para modelar a relação entre uma variável resposta (ou variável dependente) contínua e uma ou mais variáveis explicativas (ou variáveis independentes). É chamado de regressão linear simples quando se usa apenas uma variável explicativa, e regressão linear múltipla quando se usa mais de uma variável explicativa. Neste trabalho utilizaremos a regressão linear múltipla.

Os relacionamentos são modelados usando funções preditivas lineares cujos parâmetros desconhecidos do modelo são estimados a partir dos dados. Mais comumente, assume-se

que a média condicional da resposta, dados os valores das variáveis explicativas (ou preditores), seja uma função afim desses valores (função afim também é conhecida como transformação linear (Ax) seguida por uma translação ($+b$)). Como todas as formas de análise de regressão, a regressão linear se concentra na distribuição de probabilidade condicional da resposta, dados os valores dos preditores, e não na distribuição de probabilidade conjunta de todas essas variáveis.

A regressão linear foi o primeiro tipo de análise de regressão a ser estudada rigorosamente e usada extensivamente em aplicações práticas. Isso ocorre porque os modelos que dependem linearmente de seus parâmetros desconhecidos são mais fáceis de ajustar do que os modelos não linearmente relacionados aos seus parâmetros, e porque as propriedades estatísticas dos estimadores resultantes são mais fáceis de se determinar.

A regressão linear tem muitos usos práticos, o utilizado neste trabalho é o de previsão. A regressão linear pode ser usada para ajustar um modelo preditivo a um conjunto de dados observado de valores da resposta e variáveis explicativas. Depois de desenvolver esse modelo, se valores adicionais das variáveis explicativas forem coletados sem um valor de resposta definido, o modelo ajustado pode ser usado para fazer uma previsão da resposta.

Os modelos de regressão linear geralmente são ajustados usando a abordagem de mínimos quadrados, mas também podem ser ajustados de outras maneiras como minimizando uma versão penalizada da função custo de mínimos quadrados como na regressão de Ridge (penalidade de norma L2) e LASSO (penalidade de norma L1), mais informações de como funcionam tais técnicas de penalidade ver Santosa e Symes (1986) e Tibshirani (1996).

Dado um conjunto de dados $\{y_i,\, x_{i1}, \ldots, x_{ip}\}_{i=1}^{n}$ de dimensão n, um modelo de regressão linear pressupõe que a relação entre a variável dependente y e o vetor de regressores x é linear. Essa relação é modelada através de uma variável de erro ε - uma variável aleatória não observada que adiciona "ruído"à relação linear entre a variável dependente e os regressores. Assim, o modelo assume a forma:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n, \tag{3.3}$$

onde T denota a transposição, de modo que $x_i^{\mathsf{T}}\boldsymbol{\beta}$ é o produto interno entre os vetores x_i e β.

Essas equações também podem ser representadas na notação matricial, como

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{3.4}$$

onde

$$\mathrm{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \mathrm{X} = \begin{pmatrix} \mathbf{x}_1^{\mathsf{T}} \\ \mathbf{x}_2^{\mathsf{T}} \\ \vdots \\ \mathbf{x}_n^{\mathsf{T}} \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}, \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}, \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

Sendo que,

- $\mathbf{y}$ é um vetor de valores observados $y_i (i = 1, \ldots, n)$ também conhecido como variável resposta ou variável dependente. Essa variável não deve ser confundida com os valores previstos, que são indicados como $\hat{y}$. A decisão sobre qual variável em um conjunto de dados ser modelada como variável dependente e quais são modeladas como variáveis independentes baseia-se na presunção de que o valor de uma das variáveis é causado por, ou diretamente influenciado por outras variáveis.

- X pode ser visto como uma matriz de vetores de linha $\mathbf{x}_i$, conhecidos como variáveis explicativas ou variáveis independentes. Geralmente, uma constante é incluída como um dos regressores, o elemento correspondente de β é chamado de intercepto. Muitos procedimentos de inferência estatística para modelos lineares exigem a presença de um intercepto, por isso o mesmo é frequentemente incluído mesmo que considerações teóricas sugiram que seu valor seja zero.

- β é um vetor de parâmetro dimensional $(p+1)$, em que β_0 é o termo de intercepto, se houver um no modelo - caso contrário, β é p-dimensional. Seus elementos são conhecidos como efeitos ou coeficientes de regressão, embora o último termo as vezes seja reservado para os efeitos estimados. A estimativa estatística e inferência na regressão linear se concentra em β, os elementos desse vetor de parâmetro são interpretados como derivadas parciais da variável dependente em relação às várias variáveis independentes.

- ε é um vetor de valores ε_i. Essa parte do modelo é chamada de termo de erro ou, as vezes, ruído. Essa variável captura todos os outros fatores que influenciam a variável dependente y, exceto os regressores x. A relação entre o termo de erro e os regressores, por exemplo, sua correlação, é uma consideração crucial na formulação de um modelo de regressão linear, pois determinará o método de estimativa apropriado.

Uma explicação mais detalhada sobre como a regressão linear estima seus parâmetros pode ser encontrada em Freedman (2009).

Neste trabalho, o treinamento da regressão linear foi feito utilizando-se a biblioteca em Python scikit-learn (Cournapeau (2007)) que é uma das bibliotecas mais utilizadas para treinamento de modelos de aprendizado de máquina na linguagem de programação Python.

3.6 Resultados da Regressão Linear

O modelo de regressão linear é relativamente simples e de fácil interpretabilidade, uma boa estratégia de modelagem é no início do processo de modelagem realizar o treinamento do mesmo e utilizar seu desempenho como base para comparação futura com outros experimentos utilizando-se de outros modelos de aprendizado de máquina. Assim, tal modelo foi utilizado apenas para obter um desempenho base.

As estimativas para os parâmetros deste modelo estão dispostas na tabela abaixo.

Tabela 3.1: *Estimativas da regressão linear*

Variável ou Categoria	Estimativa
education_var_01_quantity_pct	0.06117
family_var_01_suitable_pct	-0.00021
family_var_01_semi_suitable_pct	0.00140
family_var_01_inappropriate_pct	-0.00055
fertility_var_01_has_children_pct	0.02855
fertility_var_01_children_born_pct	0.25068
fertility_var_01_children_borned_live_pct	-0.28298
fertility_var_01_children_borned_dead_pct	-0.00617
fertility_var_02_married_pct	0.03087
fertility_var_02_separated_pct	-0.00328
fertility_var_02_divorced_pct	0.00997
fertility_var_02_widow_pct	0.01008
fertility_var_02_single_pct	0.06214
fertility_var_03_total_pct	0.16219
work_var_01_regular_pct	0.03426
work_var_01_irregular_pct	0.04609
social_indicator_var_01_15_to_24_years_pct	-0.00144
social_indicator_var_01_25_to_59_years_pct	-0.02038
social_indicator_var_01_60_to_more_years_pct	0.00484
social_indicator_var_02_suitable_pct	-0.00088
social_indicator_var_02_semi_suitable_pct	-0.01605
social_indicator_var_02_inappropriate_pct	-0.00055
social_indicator_var_03_responsable_illiterate_pct	0.01571
social_indicator_var_03_inappropriate_residence_pct	-0.00011
social_indicator_var_03_responsable_illiterate_and_inappropriate_residence_pct	0.00063
enem_var_01_enem_score_mean_pct	0.33489
enem_var_01_enem_score_std_pct	-0.01912
enem_var_01_enem_score_median_pct	-0.34682
state_ac	-0.07552
state_al	0.01843
state_ba	0.01230
state_ce	-0.00753
state_es	0.01247
state_go	0.03169
state_ma	0.06556
state_mg	0.04068
state_ms	-0.00889
state_mt	0.00559
state_pa	0.00188
state_pb	0.02969
state_pe	0.00794
state_pi	-0.00892
state_pr	-0.00117
state_rj	-0.00751
state_rn	0.00427
state_ro	0.01943
state_rr	-0.02139
state_rs	-0.02576
state_sc	-0.01996
state_se	0.01385
state_sp	-0.02229
state_to	-0.06484

Nota-se que em alguns casos houve inversão de estimativas com relação ao esperado, como exemplo de tal inversão temos os gráficos de dispersão da variáveis explicativas `"enem_var_01_enem_score_mean_pct"` e `"enem_var_01_enem_score_median_pct"` (presentes no apêndice B.1). Tais gráficos mostram uma dispersão de valores muito próxima entre as duas variáveis, conceitualmente também é esperado que a relação com a variável resposta seja semelhante, porém as estimativas são estritamente opostas assumindo os valores 0.33489 e -0.34682, respectivamente.

O desempenho do modelo foi avaliado, conforme mencionado anteriormente, utilizando-se as medidas RMSE e bandas de acerto. Na tabela abaixo apresenta-se o RMSE sobre a base de treinamento ("Treino"), sobre a base de validação ("Validação"), sobre a base de validação utilizando-se um modelo aleatório ("Validação (modelo aleatório)") e sobre a base de validação utilizando-se um modelo nulo ("Validação (modelo nulo)"). O modelo aleatório apresenta um número aleatório que esteja entre o mínimo e o máximo observado na validação, foi criado apenas para critério de comparação com o modelo de regressão linear, assim como o modelo nulo, sendo que o modelo nulo é a média da variável resposta na base de treinamento. Nota-se um erro estritamente superior do modelo aleatório sobre o erro da regressão linear, o que implica que o modelo teve assertividade em suas predições e também que existe a possibilidade de previsão da variável resposta utilizando-se as variáveis explicativas criadas neste trabalho.

Tabela 3.2: *RMSE resultante da regressão linear*

Base	Estatística
Treino	0.0484
Validação	0.0628
Validação (modelo aleatório)	0.1868
Validação (modelo nulo)	0.0525

Na tabela abaixo apresenta-se a banda de acerto sobre a base de treinamento ("Treino"), sobre a base de validação ("Validação"), sobre a base de validação utilizando-se um modelo aleatório ("Validação (modelo aleatório)") e sobre a base de validação utilizando-se um modelo nulo ("Validação (modelo nulo)"). Nota-se que a regressão linear teve a maior concentração em "Bem Estimado"com 65% na validação, e 75% no treino.

Tabela 3.3: *Banda de acerto da regressão linear*

Base	Subestimado	Bem Estimado	Superestimado
Treino	0.1262	0.75	0.1238
Validação	0.1786	0.65	0.1714
Validação (modelo aleatório)	0.3929	0.2928	0.3143
Validação (modelo nulo)	0.1214	0.7143	0.1643

No gráfico abaixo nota-se, apesar de suave, uma relação de linearidade entre os valores preditos (eixo das ordenadas) e os valores observados (eixo das abscissas) em ambas bases de dados (treino e validação).

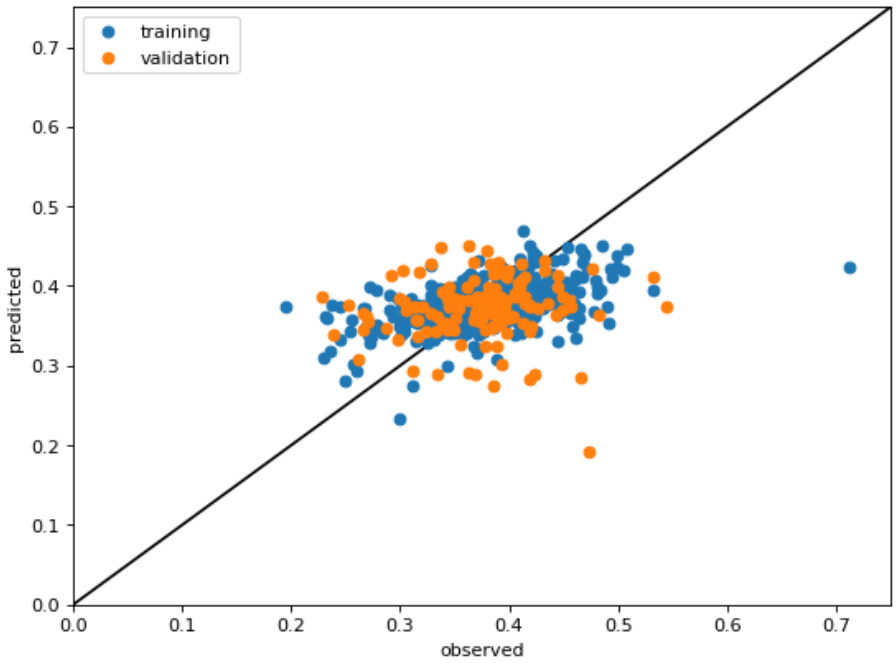


Figura 3.1: *Gráfico de dispersão da Regressão Linear*

3.7 Random Forest

O Random Forest é um método de aprendizado de máquina para classificação e regressão que opera construindo várias árvores de decisão (para classificação) ou árvores de regressão (para regressão), sendo que a predição de tal método é dada pela moda dos valores preditos em cada árvore, no caso da classificação, ou é dada pela média dos valores preditos em cada árvore, no caso de regressão. A forma de seleção aleatória das árvores reduz a incidência de overfitting no treino sobre a base de treinamento.

O Random Forest foi criado por Tin Kam Ho em Ho (1995) usando o método do subespaço aleatório, que, na formulação de Ho, é uma maneira de implementar a abordagem de "discriminação estocástica"à classificação proposta por Eugene Kleinberg (ver Kleinberg (1990)).

O algoritmo é primordialmente baseado em árvores de decisão que é um método popular para várias tarefas de aprendizado de máquina. A árvore de decisão atende aos requisitos para servir como um procedimento de prateleira para mineração de dados, porque é invariável à escala e transformações em variáveis explicativas, é robusto para inclusão de variáveis irrelevantes e seu resultado, na maioria das vezes, é interpretável.

Em particular, árvores muito profundas tendem a aprender padrões altamente irregulares: elas superestimam seus conjuntos de treinamento, ou seja, têm um viés baixo, mas uma variância muito alta (comumente conhecido como bias-variance tradeoff). O Random Forest é uma maneira de calcular a média de várias árvores de decisão, treinadas em diferentes partes do mesmo conjunto de treinamento, com o objetivo de reduzir a variância. Isso ocorre às custas de um pequeno aumento no viés e perda de interpretabilidade - uma vez que são criadas diversas árvores -, mas geralmente se aumenta muito o desempenho no modelo final.

O algoritmo de treinamento para o Random Forest aplica a técnica geral de agregação de bootstrap conhecida como Bagging. Dado um conjunto de treinamento $X = x_1, \ldots, x_n$ com respostas $Y = y_1, \ldots, y_n$, selecionados repetidamente B vezes, seleciona-se uma amos-

tra aleatória com repetição do conjunto de treinamento e se realiza o treinamento das árvores a essas amostras:

Para $b = 1, ..., B$:

1. Amostra, com repetição, n exemplos de treinamento de X, Y, chamados de X_b, Y_b.
2. Treine uma árvore de classificação ou regressão f_b em X_b, Y_b.

Após o treinamento, predições para amostras não vistas x' podem ser feitas calculando a média das predições de todas as árvores de regressão individuais em x':

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x') \tag{3.5}$$

Ou, no caso de árvores de classificação, utilizando-se o voto da maioria.

O procedimento de bootstrapping leva a um melhor desempenho do modelo pois diminui a variância do modelo, sem aumentar o viés. Isso significa que, embora as previsões de uma única árvore sejam altamente sensíveis ao ruído em seu conjunto de treinamento, a média de muitas árvores não é, desde que as árvores não estejam correlacionadas. Simplesmente treinar muitas árvores em um único conjunto de treinamento daria árvores altamente correlacionadas (ou até a mesma árvore muitas vezes, se o algoritmo de treinamento for determinístico). A amostragem por bootstrapping é uma maneira de descorrelacionar as árvores, utilizando-se diferentes conjuntos de treinamento.

O procedimento descrito acima trata do algoritmo original de bagging para árvores. O Random Forest difere apenas por uma característica específica, ele utiliza um algoritmo de aprendizado de árvore modificado que seleciona, a cada divisão do aprendizado da árvore, um subconjunto aleatório de variáveis explicativas. A motivação para tal vem da correlação das árvores em uma amostra de bootstrap comum, i.e., se uma ou algumas variáveis explicativas são preditores muito fortes para a variável de resposta, esses recursos serão selecionados em muitas das árvores B causando uma alta correlação entre as árvores treinadas.

Geralmente utiliza-se, para um problema de classificação com p variáveis explicativas, $\sqrt{p}$ (arredondado para baixo) variáveis explicativas em cada divisão. Para problemas de regressão, recomenda-se $p/3$ (arredondado para baixo). Contudo, na prática, os melhores valores para esses parâmetros dependerão do problema e devem ser tratados na otimização dos hiperparâmetros.

Neste trabalho, o treinamento do random forest foi feito utilizando-se a biblioteca em Python scikit-learn (Cournapeau (2007)) na linguagem de programação Python.

3.8 Resultados do Random Forest

O treinamento do modelo Random Forest utilizou-se da metodologia explicada na Seção 3.2 e os hiperparâmetros testados foram:

- `max_depth`: Profundidade máxima da árvore.
- `max_features`: Número de variáveis explicativas a serem consideradas ao procurar a melhor divisão da árvore.

- `min_samples_split`: Número mínimo de amostras necessárias para se dividir um nó da árvore.
- `min_samples_leaf`: Número mínimo de amostras necessárias para se deixar em uma folha da árvore.
- `n_estimators`: Número de árvores na floresta.
- `bootstrap`: Se amostras de bootstrap são usadas na construção das árvores.
- `criterion`: Função a ser usada para medir a qualidade da divisão da árvore (erro quadrático médio, erro absoluto médio, etc.).

O número de iterações no Randomized Search a serem realizadas foi de 300 com um método de validação cruzada 5-fold (Kohavi (1995)). Por fim, o modelo final após a otimização do Randomized Search possui os hiperparâmetros apresentados na tabela abaixo.

Tabela 3.4: *Hiperparâmetros finais do Random Forest*

Hiperparâmetro	Valor
`max_depth`	10
`max_features`	13
`min_samples_split`	0.4774005510811298
`min_samples_leaf`	0.05170415106602393
`n_estimators`	75
`bootstrap`	False
`criterion`	mse

O modelo Random Forest não é de fácil interpretabilidade, apesar de ser possível ver e analisar a estrutura de cada uma das árvores separadamente que compõem o mesmo, não é direta a interpretação de como foi feita a predição dos valores por tal algoritmo. Assim, para interpretar como cada variável afeta o modelo e avaliar a importância de cada uma delas utilizamos o método de Shapley Values (descrito na Seção 3.3), sendo que o gráfico resultante se encontra abaixo.

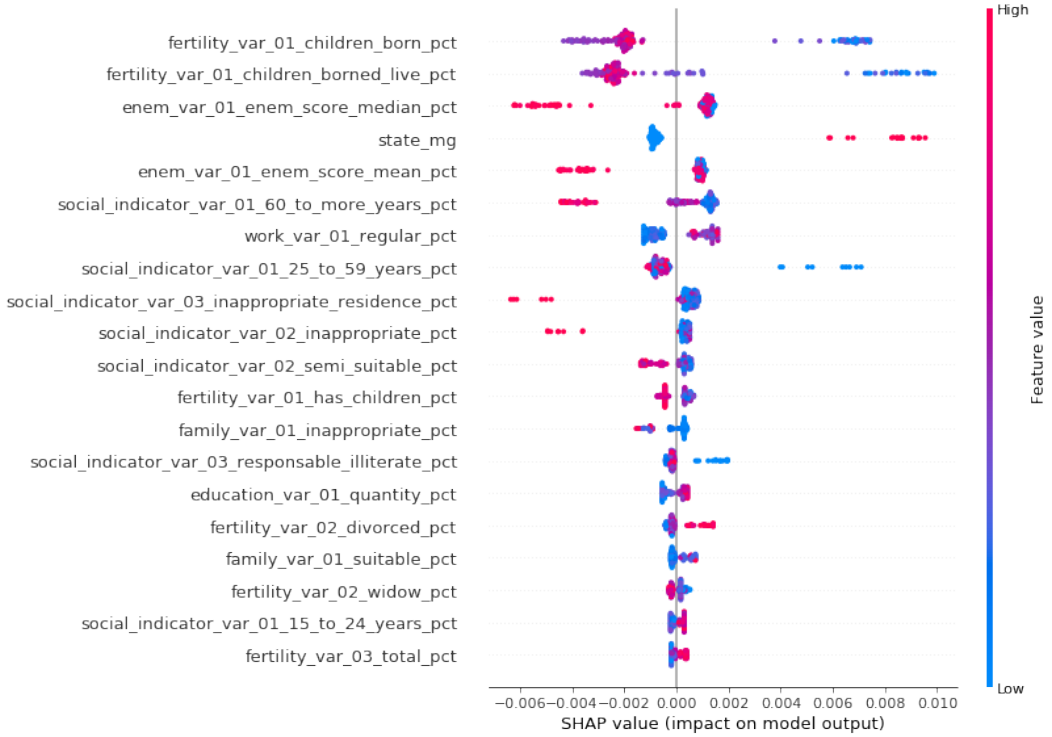


Figura 3.2: *Shapley value do Random Forest*

No gráfico, as variáveis apresentadas à esquerda estão ordenadas de forma decrescente em nível de importância no modelo, no eixo das ordenadas se apresentam os valores de Shapley Values - que pode ser lido como a influência no valor de predição do modelo, tanto positivo quanto negativo -, e as cores como o valor da variável explicativa propriamente dita, sendo vermelho o maior valor possível e azul o menor valor possível para a variável em questão. Então, no caso da variável `fertility_var_01_children_born_pct` - que se trata do percentual de crianças nascidas por município em 2000 dividido pela mesma informação em 2010 -, a interpretação se dá na forma de que quanto menor o valor de tal variável, mais positiva será a influência sobre a predição de polaridade negativa no relatório do município em questão.

Assim como a regressão linear, o desempenho do modelo foi avaliado utilizando-se as medidas RMSE e bandas de acerto. Na tabela abaixo apresenta-se o RMSE sobre a base de treinamento ("Treino"), sobre a base de validação ("Validação"), sobre a base de validação utilizando-se um modelo aleatório ("Validação (modelo aleatório)") e sobre a base de validação utilizando-se um modelo nulo ("Validação (modelo nulo)"). O modelo aleatório apresenta um número aleatório que esteja entre o mínimo e o máximo observado na validação, foi criado apenas para critério de comparação com o modelo Random Forest, assim como o modelo nulo, sendo que o modelo nulo é a média da variável resposta na base de treinamento. Nota-se um erro estritamente superior do modelo aleatório sobre o erro do Random Forest.

Tabela 3.5: *RMSE resultante do Random Forest*

Base	Estatística
Treino	0.0513
Validação	0.0527
Validação (modelo aleatório)	0.1728
Validação (modelo nulo)	0.0525

Na tabela abaixo apresenta-se a banda de acerto sobre a base de treinamento ("Treino"), sobre a base de validação ("Validação"), sobre a base de validação utilizando-se um modelo aleatório ("Validação (modelo aleatório)") e sobre a base de validação utilizando-se um modelo nulo ("Validação (modelo nulo)"). Nota-se que o Random Forest teve a maior concentração em "Bem Estimado"com mais de 70% na validação e treino.

Tabela 3.6: *Banda de acerto do Random Forest*

Base	Subestimado	Bem Estimado	Superestimado
Treino	0.1333	0.731	0.1357
Validação	0.1357	0.7286	0.1357
Validação (modelo aleatório)	0.5214	0.1215	0.3571
Validação (modelo nulo)	0.1214	0.7143	0.1643

No gráfico abaixo nota-se, apesar de suave, uma relação de linearidade entre os valores preditos (eixo das ordenadas) e os valores observados (eixo das abscissas) em ambas bases de dados (treino e validação).

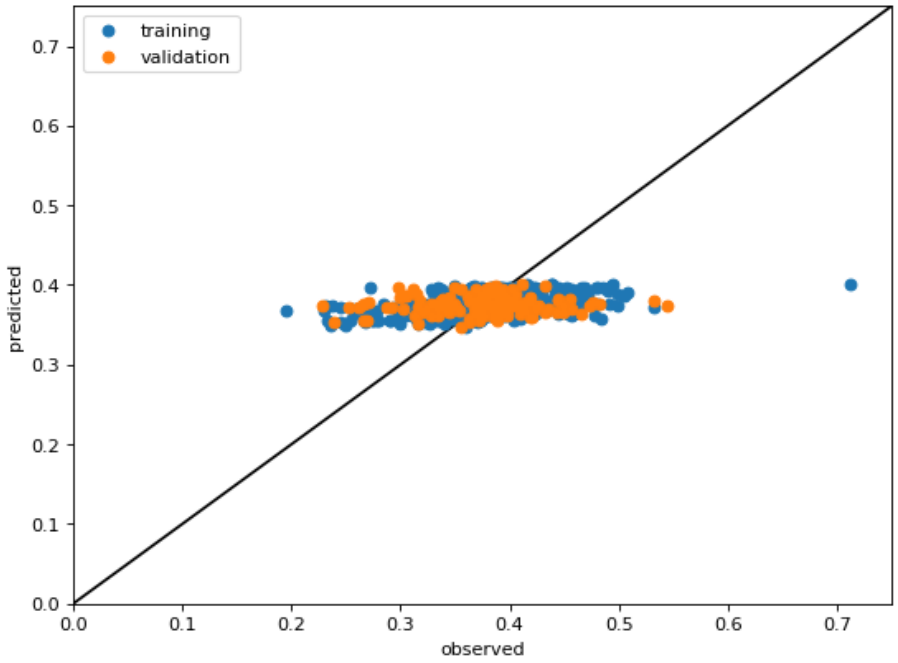


Figura 3.3: *Gráfico de dispersão do Random Forest*

3.9 XGBoost

O XGBoost (eXtreme Gradient Boosting) é uma implementação do algoritmo Gradient Boosting que é considerado, muitas vezes, melhor que o Gradient Boosting por sua capacidade em reduzir a chance de overfitting. A principal diferença consiste que o XGBoost utiliza uma formalização de modelo mais regularizada, que resulta muitas vezes na redução de overfitting e melhor desempenho.
O nome XGBoost se refere ao objetivo de aumentar o limite de recursos de computação para algoritmos de árvore aprimorada, que é o motivo pelo qual muitas pessoas usam o XGBoost.
O Gradient Boosting é uma técnica de aprendizado de máquina para problemas de regressão e classificação, o algoritmo produz um modelo de previsão na forma de um conjunto de modelos de preditores fracos que são árvores de decisão. A construção do modelo é feita em etapas, como outros métodos de reforço, e os generaliza permitindo a otimização de uma função de perda diferenciável arbitrária. O Gradient Boosting teve origem quando Leo Breiman (ver Breiman (1997)) observou que boosting pode ser interpretado como um algoritmo de otimização em uma função de custo. Sendo que o algoritmo em si foi desenvolvido posteriormente por Jerome H. Friedman (ver Friedman (1999a) e Friedman (1999b)).
O Gradient Boosting combina preditores fracos em um único preditor forte de maneira iterativa. No exemplo de regressão, na otimização de mínimos quadrados, onde o objetivo é encontrar um modelo F para prever valores da forma $\hat{y} = F(x)$ minimizando o erro quadrático médio $\frac{1}{n}\sum_i(\hat{y}_i - y_i)^2$, onde i é o índice sobre algum conjunto de treinamento de tamanho n dos valores reais da variável resposta y.
Em cada etapa m, $1 \leq m \leq M$ - onde M é a quantidade de iterações -, do Gradient Boosting, supõe-se que exista algum modelo imperfeito F_m (no início, pode ser usado um modelo muito fraco que prevê apenas a média de y no conjunto de treinamento). O Gradient Boosting aprimora o F_m construindo um novo modelo que adiciona um estimador h para fornecer um modelo melhor: $F_{m+1}(x) = F_m(x) + h(x)$. Para estimar h, o algoritmo começa com a premissa de que um h perfeito implicaria em

$$F_{m+1}(x) = F_m(x) + h(x) = y \tag{3.6}$$

ou, de forma equivalente,

$$h(x) = y - F_m(x) \tag{3.7}$$

Portanto, o Gradient Boosting ajustará h ao $y - F_m(x)$ e cada F_{m+1} tenta corrigir os erros de seu predecessor F_m, utilizando a função custo L. Uma generalização dessa idéia para funções de perda que não sejam erros quadráticos e para problemas de classificação, resulta da observação de que os resíduos $y - F(x)$ para um determinado modelo são os gradientes negativos (em relação à $F(x)$) da função de perda de erro ao quadrado $\frac{1}{2}(y - F(x))^2$.
Para uma melhor exemplificação, a seguir foi extraído do livro Hastie e Friedman (2001) o algoritmo do Gradient Boosting e explicado detalhadamente cada passo do mesmo.

1. Inicialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. Para m = 1 até M:

(a) Para $i = 1, 2, ..., N$ calcule:

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}} \tag{3.8}$$

(b) Treine uma árvore de regressão sobre a variável resposta resíduo r_{im} resultando nas regiões R_{jm}, $j = 1, 2, ..., J_m$.

(c) Para $j = 1, 2, ..., J_m$ calcule:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma) \tag{3.9}$$

(d) Atualize $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

3. Resultado é $\hat{f}(x) = f_M(x)$

Sendo que,

- Em 1., inicializa-se a primeira iteração ou árvore (f_0, ou seja, $m = 0$).
- Em 2., o valor M representa o total de iterações (ou árvores).
- Em 2.(a), para cada observação i - de um total de N - calcula-se o resíduo dado pelo negativo do gradiente da função perda avaliado em f_{m-1}, por exemplo, no caso de regressão o resultante do gradiente poderia ser $y_i - f(x_i)$ resultante da função perda $\frac{1}{2}(y - F(x))^2$.
- Em 2.(b), treina-se uma árvore de regressão sobre a variável resposta resíduo, sendo que J_m seria o número total de folhas da referida iteração.
- Em 2.(c), calcula-se o argumento que minimiza a função perda para cada folha da referida iteração sobre a predição de f_{m-1}.
- Em 2.(d), atualiza-se a função $f_m(x)$ utilizando-se a função resultante da iteração anterior ($f_{m-1}(x)$) e o argumento calculado no item anterior (γ_{jm}) aplicado apenas aos casos em que o registro pertence à folha em questão ($x \in R_{jm}$).
- O resultado final se dá pela função da última iteração (M).

No algoritmo XGBoost, o tamanho da árvore e a magnitude dos pesos são controlados por parâmetros com regularização padrão. Uma infinidade de parâmetros é usada ainda para controlar o tamanho e a forma das árvores. A regularização de tais parâmetros, no entanto, provou ser muito poderosa e fez o algoritmo se tornar muito robusto, sendo que, atualmente, é um dos mais utilizados em competições de aprendizado de máquina.
Neste trabalho, o treinamento do XGBoost foi feito utilizando-se a biblioteca em Python xgboost (Chen (2014)) na linguagem de programação Python.

3.10 Resultados do XGBoost

O treinamento do modelo XGBoost utilizou-se da metodologia explicada na Seção 3.2 e os hiperparâmetros testados foram:

- `max_depth`: Profundidade máxima de uma árvore. Aumentar esse valor tornará o modelo mais complexo e mais propenso à overfitting.
- `learning_rate`: Define o tamanho da etapa na otimização do algoritmo, usado na atualização dos estimadores servindo para controlar o ajuste a cada etapa.
- `colsample_bytree`: Taxa de subamostra de variáveis explicativas ao construir cada árvore. A subamostragem ocorre uma vez para cada árvore construída.
- `colsample_bylevel`: Taxa de subamostra de variáveis explicativas para cada nível de profundidade. A subamostragem ocorre uma vez para cada novo nível de profundidade alcançado em uma árvore.
- `min_child_weight`: Soma mínima do peso do nó necessária em cada árvore.
- `gamma`: Valor mínimo para redução da função perda necessária para fazer uma partição adicional em um nó da árvore. Quanto maior, mais conservador será o algoritmo.
- `reg_lambda`: Peso dos termos de regularização L2. Aumentar esse valor tornará o modelo mais conservador.
- `n_estimators`: Número de estimadores ou árvores construídas no treinamento.
- `eval_metric`: Função a ser usada para medir a qualidade da divisão da árvore (erro quadrático médio, erro absoluto médio, etc.).

O número de iterações no Randomized Search a serem realizadas foi de 300 com um método de validação cruzada 5-fold (Kohavi (1995)). Por fim, o modelo final após a otimização do Randomized Search possui os hiperparâmetros apresentados na tabela abaixo.

Tabela 3.7: *Hiperparâmetros finais do Xgboost*

Hiperparâmetro	Valor
`max_depth`	12
`learning_rate`	0.1
`colsample_bytree`	0.4
`colsample_bylevel`	0.7
`min_child_weight`	7.0
`gamma`	0
`reg_lambda`	100.0
`n_estimators`	50
`eval_metric`	rmse

O modelo XGBoost não é de fácil interpretabilidade, apesar de ser possível ver e analisar a estrutura de cada uma das árvores separadamente que compõem o mesmo, não é direta a interpretação de como foi feita a predição dos valores por tal algoritmo. Assim, para

interpretar como cada variável afeta o modelo e avaliar a importância de cada uma delas utilizamos o método de Shapley Values (descrito na Seção 3.3) e o gráfico resultante se encontra abaixo.

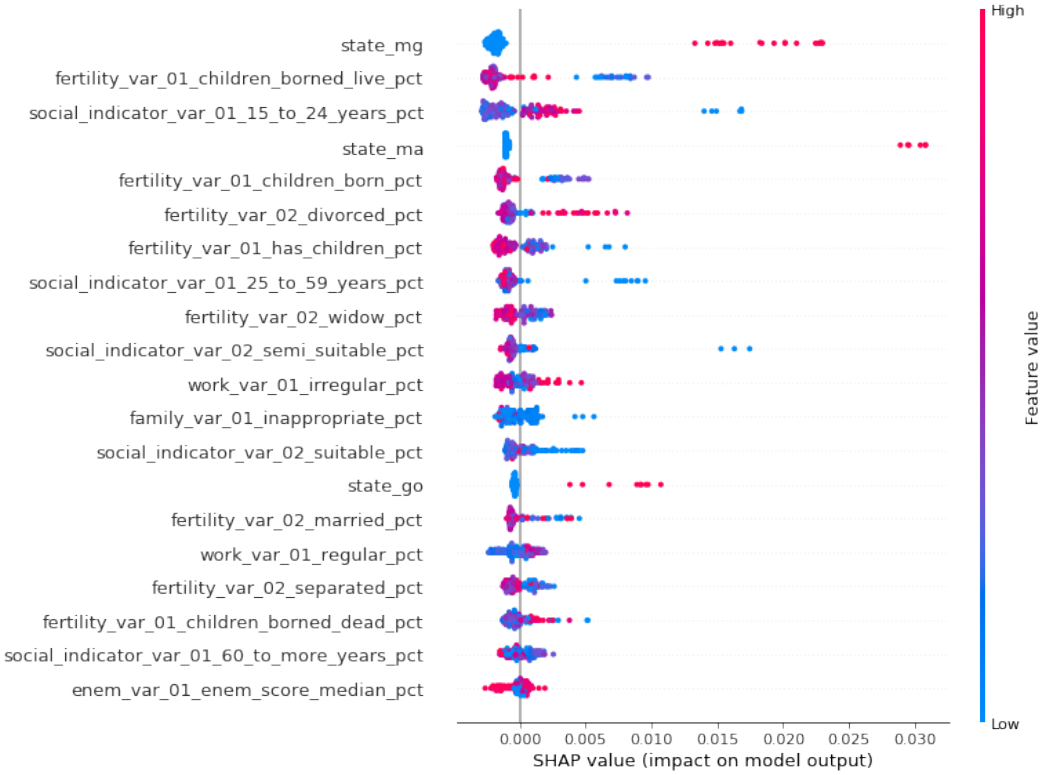


Figura 3.4: *Shapley value do Xgboost*

No gráfico, as variáveis apresentadas à esquerda estão ordenadas de forma decrescente em nível de importância no modelo, no eixo das ordenadas se apresentam os valores de Shapley Values - que pode ser lido como a influência no valor de predição do modelo, tanto positivo quanto negativo -, e as cores como o valor da variável explicativa propriamente dita, sendo vermelho o maior valor possível e azul o menor valor possível para a variável em questão. Então, no caso da variável `state_mg` - que se trata do estado de origem do município avaliado -, a interpretação se dá na forma de que quanto maior o valor de tal variável (neste caso um, por se tratar de uma variável binária), mais positiva será a influência sobre a predição de polaridade negativa no relatório do município em questão. Assim como a regressão linear, o desempenho do modelo foi avaliado utilizando-se as medidas RMSE e bandas de acerto. Na tabela abaixo apresenta-se o RMSE sobre a base de treinamento ("Treino"), sobre a base de validação ("Validação"), sobre a base de validação utilizando-se um modelo aleatório ("Validação (modelo aleatório)") e sobre a base de validação utilizando-se um modelo nulo ("Validação (modelo nulo)"). O modelo aleatório apresenta um número aleatório que esteja entre o mínimo e o máximo observado na validação, foi criado apenas para critério de comparação com o modelo XGBoost, assim como o modelo nulo, sendo que o modelo nulo é a média da variável resposta na base de treinamento. Nota-se um erro estritamente superior do modelo aleatório sobre o erro do XGBoost.

Tabela 3.8: *RMSE resultante do Xgboost*

Base	Estatística
Treino	0.0448
Validação	0.0521
Validação (modelo aleatório)	0.1814
Validação (modelo nulo)	0.0525

Na tabela abaixo apresenta-se a banda de acerto sobre a base de treinamento ("Treino"), sobre a base de validação ("Validação"), sobre a base de validação utilizando-se um modelo aleatório ("Validação (modelo aleatório)") e sobre a base de validação utilizando-se um modelo nulo ("Validação (modelo nulo)"). Nota-se que o XGBoost teve a maior concentração em "Bem Estimado"com mais de 70% na validação e quase 80% no treino.

Tabela 3.9: *Banda de acerto do Xgboost*

Base	Subestimado	Bem Estimado	Superestimado
Treino	0.0929	0.7976	0.1095
Validação	0.1071	0.7215	0.1714
Validação (modelo aleatório)	0.4429	0.1642	0.3929
Validação (modelo nulo)	0.1214	0.7143	0.1643

No gráfico abaixo nota-se, apesar de suave, uma relação de linearidade entre os valores preditos (eixo das ordenadas) e os valores observados (eixo das abscissas) em ambas bases de dados (treino e validação).

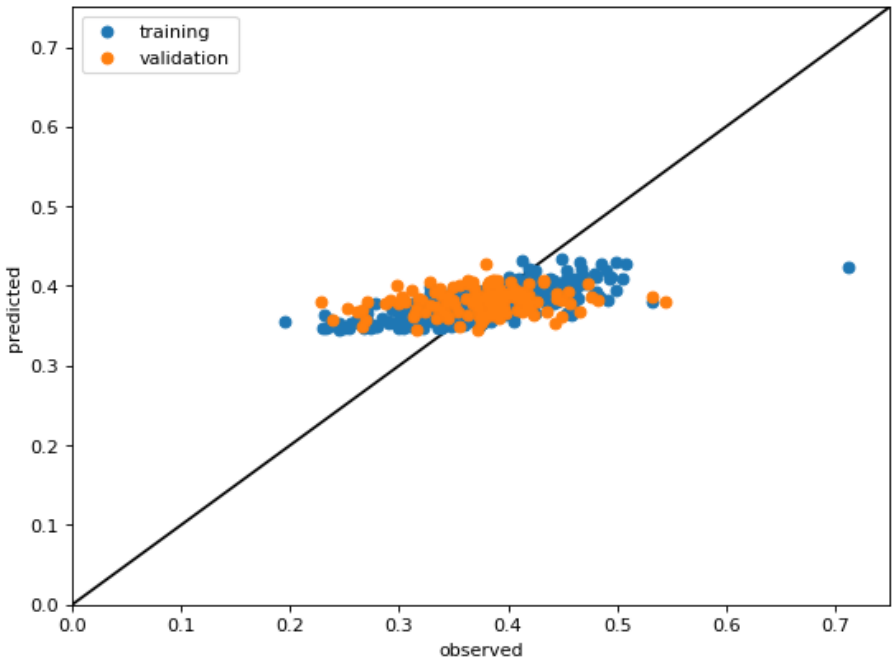


Figura 3.5: *Gráfico de dispersão do Xgboost*

Capítulo 4

Conclusões

Neste trabalho foi desenvolvido uma forma automatiza de captura de informações governamentais geopolíticas com extração de palavras de documentos, assim, garantindo a replicabilidade do trabalho. Estudamos a estrutura da informação existente em relatórios de auditoria da Controladoria-Geral da União (CGU) que se mostrou uma fonte de dados ampla, pouco estudada na literatura e que pode ser explorada por diversas perspectivas distintas podendo agregar muito valor em estudos acadêmicos. Estudamos as informações socioeconômicas do censo do Instituto Brasileiro de Geografia e Estatística (IBGE) visando encontrar informações que ajudassem na predição das informações extraídas dos relatórios de auditoria da CGU. E utilizamos, sempre que possível, metodologias aplicadas na atualidade nos âmbitos de organização (para produção deste trabalho), padronização dos códigos desenvolvidos, treinamento dos modelos de aprendizado de máquina, e visualização dos resultados (para melhor interpretação de modelos complexos pouco interpretáveis).
A extração dos dados realizada neste trabalho pode ser considerada de grande valor uma vez que, em projetos de ciência de dados, tal etapa é onerosa em termos de tempo e criatividade na criação de variáveis, além do fato de que tal fonte de dados nunca foi explorada na literatura de forma quantitativa como executado neste trabalho. Os modelos treinados sobre a base de dados foram Regressão Linear, Random Forest e XGBoost, que, apesar de não terem apresentado um desempenho excelente, mesmo utilizando-se de técnicas de otimização dos hiperparâmetros, mostraram a possibilidade da modelagem de tal problema, sendo que o modelo que apresentou o melhor desempenho foi o XGBoost.
A seguir apresentamos algumas possibilidades de pesquisas futuras relacionadas aos relatórios de auditoria da CGU.

- Utilizar outras informações disponíveis na base de dados do censo do IBGE, como informações que em primeiro momento não estejam diretamente relacionadas com a variável resposta criada a partir dos relatórios de auditoria da CGU.

- Extrair informações dos relatórios e utilizá-los como variáveis explicativas para predição, por exemplo, de variação de desempenho escolar nos municípios em questão.

- Produzir um modelo de processamento de linguagem natural sobre os relatórios de auditoria da CGU para calcular a probabilidade de percentual de palavras negativas apenas pelo contexto do relatório, utilizando as definições estabelecidas na base de dados SentiLex (Silva e Carvalho (2012)).

Apêndice A

Códigos em Python

A.1 Import reports

```
from selenium import webdriver
from selenium.webdriver.support.ui import Select
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.firefox.firefox_profile import
   FirefoxProfile

import os
import time
import codecs
```

Set configurations (only for 'edicoes_anteriores', others are manually)

```
# set driver for browser connection
DRIVER_BIN = "/Users/lucas.bruscato/Google Drive/Github-
   Lucas/master/0_drivers/geckodriver_mac"
# DRIVER_BIN = "C:\\Users\\lbruscato\\Dropbox\\Github-Lucas
   \\master\\drivers\\geckodriver_windows.exe"

# set initial folder
folder = 'edicoes_anteriores/<folder name here>'
```

Capture search dates and cities' names

```
# open link file for the chosen folder
link_file = codecs.open(os.getcwd() + "/" + folder + "/link.
   txt", 'r', "utf-8")
text_link_file = link_file.read().split('\n')
```

```
# create the initial and final date to search
initial_date = (text_link_file[1].split('-'))[0]
final_date = (text_link_file[1].split('-'))[1]

# create an empty list of cities
cities = []

# fill cities list
for i in range(2, len(text_link_file)):
    city = text_link_file[i].split('-')
    cities.append(city[1].strip())

print("initial_date = " + initial_date)
print("final_date = " + final_date)
print(cities)
```

Open browser and download all cities' report

```
# set browser preferences and profile (auto download)
profile = FirefoxProfile()
profile.set_preference('browser.helperApps.neverAsk.
   saveToDisk', "application/pdf,application/zip")
profile.set_preference('browser.download.folderList', 2)
profile.set_preference('browser.download.dir', os.getcwd() +
    "/" + folder)

# open browser
browser = webdriver.Firefox(executable_path = DRIVER_BIN,
   firefox_profile = profile)
browser.maximize_window()

for j in range(0, len(cities), 5):
    browser.get('https://auditoria.cgu.gov.br/')

    # fill fields
    Select(browser.find_element_by_id("linhaAtuacao")).
       select_by_visible_text('Fiscaliza o em Entes
       Federativos - Munic pios')

    browser.find_element_by_id("de").send_keys(initialDate,
       Keys.TAB)
    browser.find_element_by_id("ate").send_keys(finalDate,
       Keys.TAB)

    for i in range(j, j + 5):
        if (i < len(cities)):
```

```
            browser.find_element_by_id("palavraChave").
               send_keys(cities[i], Keys.COMMAND, 'a')
            browser.find_element_by_id("palavraChave").
               send_keys(Keys.COMMAND, 'x')
            browser.find_element_by_id("token-input-
               municipios").send_keys(Keys.COMMAND, 'v')
            time.sleep(2)

            browser.find_element_by_id("token-input-
               municipios").send_keys(Keys.ENTER)
            time.sleep(1)

    # search and download files
    browser.find_element_by_id("btnPesquisar").click()
    time.sleep(2)

    browser.find_element_by_id("btnSelectAll").click()
    time.sleep(1)

    browser.find_element_by_id("btnBaixar").click()
    time.sleep(1)
```

A.2 Rename Folders

```
import os
import os.path
import random
```

Create list of folders to access and rename files

```
folders = ["2000/education",
           "2000/family",
           "2000/fertility",
           "2000/social_indicator",
           "2000/work",
           "2010/education",
           "2010/family",
           "2010/fertility",
           "2010/social_indicator",
           "2010/work"]

state_acronym_and_name_list = [
    ["ac", "acre"],
    ["al", "alagoas"],
```

```
    ["ap", "amapa"],
    ["am", "amazonas"],
    ["ba", "bahia"],
    ["ce", "ceara"],
    ["df", "distrito_federal"],
    ["es", "espirito_santo"],
    ["go", "goias"],
    ["ma", "maranhao"],
    ["mt", "mato_grosso"],
    ["ms", "mato_grosso_do_sul"],
    ["mg", "minas_gerais"],
    ["pa", "para"],
    ["pb", "paraiba"],
    ["pr", "parana"],
    ["pe", "pernambuco"],
    ["pi", "piaui"],
    ["rj", "rio_de_janeiro"],
    ["rn", "rio_grande_do_norte"],
    ["rs", "rio_grande_do_sul"],
    ["ro", "rondonia"],
    ["rr", "roraima"],
    ["sc", "santa_catarina"],
    ["sp", "sao_paulo"],
    ["se", "sergipe"],
    ["to", "tocantins"]]
```

Rename files based on pattern

```
for folder in folders:
    print(folder)
    for inner_folder in os.listdir(folder):
        new_folder_name = inner_folder
        for acronym in state_acronym_and_name_list:
            if inner_folder == acronym[0]:
                new_folder_name = acronym[1]
                break
        new_folder_name = str.replace(new_folder_name, "
           _munic_xls", "")
        new_folder_name = str.replace(new_folder_name, "_xls
           ", "")

        print(folder + "/" + inner_folder + " -> " + folder
           + "/" + new_folder_name)

        os.rename(folder + "/" + inner_folder,
                  folder + "/" + new_folder_name)
```

A.3 Extract Explanatory Features

```
import os
import shutil
import pandas as pd
```

Script to select needed values for building features

```
def select_feature_range(df, columns_index, initial_row):
    size = 0
    i = 0

    if initial_row != 0:
        initial_row -= 2
        size += 1

    for identifier in df.iloc[initial_row:, columns_index
       [0]]:
        i += 1
        if identifier == 'Municípios':
            initial_row = i

        if initial_row != 0:
            size += 1
            if type(identifier) == float or identifier[:6]
               == 'Fonte:':
                break

    return df.iloc[initial_row:(initial_row+size-2),
       columns_index]
```

```
def create_feature_files(year, subject,
   feature_identification, columns_index,
                            name_str_identification_prefix,
                               name_str_identification_suffix,
                            number_of_character_prefix,
                               number_of_character_suffix,
                            col_names, path, states, is_to_copy
                               , initial_row):
    quantity_of_features_created = 0

    for state in states:

        if not state.startswith('.') and is_to_copy:
```

```
            files = os.listdir(path + state)

            for file in files:
                if ((name_str_identification_prefix == None
                   or
                    file[:number_of_character_prefix] ==
                       name_str_identification_prefix) and
                    (file[len(file)-
                       number_of_character_suffix:] ==
                       name_str_identification_suffix)):

                    df = pd.read_excel(path + state + '/' +
                       file)

                    df_var = select_feature_range(df,
                       columns_index, initial_row)
                    df_var.columns = col_names

                    # remove accents and lower case
                    df_var.city = df_var.city.str.normalize(
                       'NFKD').str.encode('ascii', errors='
                       ignore').\
                        str.decode('utf-8').str.lower()

                    # save in a csv file
                    df_var.to_csv(path + state + '/' + year
                       + '_' + subject + '_' + state + '
                       _var_' +
                                     feature_identification
                                        + '.csv',
                                  index=False,
                                  sep = ',')

                    print(path + state + '/' + year + '_' +
                       subject + '_' + state + '_var_' +
                              feature_identification + '.csv
                                 ')
                    quantity_of_features_created += 1

    print('Quantity of features created: ' + str(
       quantity_of_features_created))
```

For security reasons, change parameter 'is_to_copy' to True before start running

```
# set parameter here #
is_to_copy = False
```

```
######################
```

Set initial parameters to select needed values for building *each* features

```
# set parameters here #
year = '2010'
subject = 'family'
feature_identification = '02'
columns_index = [0, 4]
name_str_identification_prefix = 'tab4_' # String or None if
    don't need a suffix to search
name_str_identification_suffix = '2_1.xls'
col_names = ['city',
             'qt']
initial_row = 0 # integer or 0 if it follows the structure (
   python pattern)
######################

number_of_character_suffix = len(
   name_str_identification_suffix)
path = year + '/' + subject + '/'
states = os.listdir(path)

print(path)
print(feature_identification)

if name_str_identification_prefix != None:
    number_of_character_prefix = len(
       name_str_identification_prefix)
    print(name_str_identification_prefix + '-*-' +
       name_str_identification_suffix)
else:
    number_of_character_prefix = 0
    print('-*-' + name_str_identification_suffix)
```

```
create_feature_files(year, subject, feature_identification,
   columns_index,
                     name_str_identification_prefix,
                        name_str_identification_suffix,
                     number_of_character_prefix,
                        number_of_character_suffix,
                     col_names, path, states, is_to_copy,
                        initial_row)
```

A.4 Extract Explanatory Features Enem

```
import os
import shutil
import pandas as pd
```

Extract features from enem 2000

```
columns_to_select = ['NO_MUNICIPIO_RESIDENCIA',
                     'SG_UF_RESIDENCIA',
                     'TP_PRESENCA',
                     'TP_STATUS_REDACAO',
                     'NU_NOTA_OBJETIVA',
                     'NU_NOTA_REDACAO']
```

```
df = pd.read_csv('2000/Dados/MICRODADOS_ENEM_2000.csv',
                 sep=';',
                 usecols=columns_to_select)

df = df.query('TP_PRESENCA == 1 and TP_STATUS_REDACAO == "P"
   ')
```

```
df['city_state'] = (df.NO_MUNICIPIO_RESIDENCIA + '_' + df.
   SG_UF_RESIDENCIA).str.normalize('NFKD')\
                    .str.encode('ascii', errors='ignore')\
                    .str.decode('utf-8').str.lower()

df['enem_score'] = (df['NU_NOTA_OBJETIVA']+df['
   NU_NOTA_REDACAO'])/2
```

```
df_grouped = df[['city_state', 'enem_score']].groupby('
   city_state').agg(['mean', 'std', 'median'])

df_grouped.columns = ['enem_score_mean', 'enem_score_std', '
   enem_score_median']
```

```
df_grouped.to_csv('2000/2000_enem_score_var_01.csv', sep=';'
   )
```

Extract features from enem 2010

```
columns_to_select = ['NO_MUNICIPIO_RESIDENCIA',
                     'SG_UF_RESIDENCIA',
                     'TP_PRESENCA_CN',
                     'TP_PRESENCA_CH',
                     'TP_PRESENCA_LC',
                     'TP_PRESENCA_MT',
                     'NU_NOTA_CN',
                     'NU_NOTA_CH',
                     'NU_NOTA_LC',
                     'NU_NOTA_MT',
                     'TP_STATUS_REDACAO',
                     'NU_NOTA_REDACAO']
```

```
df = pd.read_csv('2010/Dados/MICRODADOS_ENEM_2010.csv',
                 sep=';',
                 usecols=columns_to_select)

df = df.query('TP_PRESENCA_CN == 1 and TP_PRESENCA_CH == 1
   and TP_PRESENCA_LC == 1 and TP_PRESENCA_MT == 1 and
   TP_STATUS_REDACAO == "P"')
```

```
df['city_state'] = (df.NO_MUNICIPIO_RESIDENCIA + '_' + df.
   SG_UF_RESIDENCIA).str.normalize('NFKD')\
                    .str.encode('ascii', errors='ignore')\
                    .str.decode('utf-8').str.lower()

df['enem_score'] = (df['NU_NOTA_CN']+df['NU_NOTA_CH']+df['
   NU_NOTA_LC']+df['NU_NOTA_MT']+df['NU_NOTA_REDACAO'])/5
```

```
df_grouped = df[['city_state', 'enem_score']].groupby('
   city_state').agg(['mean', 'std', 'median'])

df_grouped.columns = ['enem_score_mean', 'enem_score_std', '
   enem_score_median']
```

```
df_grouped.to_csv('2010/2010_enem_score_var_01.csv', sep=';'
   )
```

A.5 Create Initial SentiLex Database and Improve it

```
import csv
import pandas as pd
import os
import os.path
import random
import PyPDF2
import unidecode
import pandas as pd
from collections import Counter
import csv
```

Read SentiLex-PT02 and extract polarity

```
# read csv file
sentilex_database = pd.read_csv("SentiLex-flex-PT02.txt",
   header = None)
sentilex_database.columns = ["adjective", "description"]

# extract "polarity" from "description"
polarity = pd.DataFrame(sentilex_database.description.str.
   split('\;+').str[3].str.split('\=+').str[1])
sentilex_database = pd.concat([sentilex_database, polarity],
    axis = 1, join = 'outer')

# remove duplicates
sentilex_database = sentilex_database.iloc[:, [0, 2]].
   drop_duplicates()
sentilex_database.columns = ["adjective", "polarity"]

# select only polarities in [-1, 0, 1]
polarities = ["-1", "0", "1"]
sentilex_database = sentilex_database[sentilex_database.
   polarity.isin(polarities)]
```

```
sentilex_database.head()
```

Save initial sentilex database

```
sentilex_database.to_csv("99_01_sentilex_database.csv",
                         sep = ';',
                         encoding = 'utf-8',
                         index = False)
```

Define randomly reports for improving SentiLex-PT02

```
folders = ["ciclo_3",
           "ciclo_4",
           "ciclo_5",
           "edicoes_anteriores/sorteio_34",
           "edicoes_anteriores/sorteio_35",
           "edicoes_anteriores/sorteio_36",
           "edicoes_anteriores/sorteio_37",
           "edicoes_anteriores/sorteio_38",
           "edicoes_anteriores/sorteio_39",
           "edicoes_anteriores/sorteio_40"]

file_names_and_paths = []

for folder in folders:
    directory = '../
       programa_de_fiscalizacao_em_entes_federativos/' +
       folder

    number_of_files = len([name for name in os.listdir(
       directory) if os.path.isfile(os.path.join(directory,
       name))]) - 3
    random.seed(7)
    random_file_number = int(random.uniform(0,
       number_of_files))

    file_name_and_path = directory + "/" + os.listdir(
       directory)[random_file_number]
    file_names_and_paths.append(file_name_and_path)

file_names_and_paths
```

Import reports, collect unique words and save words not in SentiLex-PT02

```
print("List of reports read to improve SentiLex database")

words_without_polarity_full = pd.DataFrame(columns=['
   adjective', 'polarity'])

for file_number in range(0, len(file_names_and_paths)):

    file_name = file_names_and_paths[file_number]
    print(file_name)

    # create a pdf object
```

```
    file = open(file_name, 'rb')

    # create a pdf reader object
    file_reader = PyPDF2.PdfFileReader(file)

    # iterate all documents
    word_index = -1
    flag_in_a_word = 0
    words = []

    for i in range(file_reader.numPages):
        page = unidecode.unidecode(file_reader.getPage(i).
           extractText().lower())

        for j in range(len(page)):
            letter = page[j]

            if (not letter.isalpha()) and flag_in_a_word !=
               0:
                flag_in_a_word = 0
            elif letter.isalpha() and flag_in_a_word == 0:
                flag_in_a_word = 1
                word_index += 1
                words.append(letter)
            elif letter.isalpha() and flag_in_a_word != 0:
                words[word_index] += letter

    words_unique = pd.DataFrame(pd.DataFrame(words).iloc[:,
       0].unique())
    words_unique.columns = ["adjective"]

    words_with_polarity = words_unique.merge(
        sentilex_database,
        left_on="adjective",
        right_on="adjective",
        how="left")

    words_without_polarity_full = pd.concat(
        [words_without_polarity_full, words_with_polarity[
           words_with_polarity.polarity.isnull()]]
    )


words_without_polarity_full = pd.DataFrame(
   words_without_polarity_full.adjective.unique())
words_without_polarity_full.columns = ['adjective']
words_without_polarity_full['polarity'] = ''
```

```
words_without_polarity_full.sort_values(by=['adjective'],
   inplace=True)

words_without_polarity_full.to_csv("improving_sentilex/99
   _create_improving_sentilex.csv",
                                      sep=';',
                                      encoding='utf-8',
                                      index=False)
```

A.6 Create Target Feature

```
import PyPDF2
import unidecode
import pandas as pd
from collections import Counter
import csv
import os
import os.path
import random
import re
import datetime
```

Create lists with all paths to all files

```
folders = ["ciclo_3",
           "ciclo_4",
           "ciclo_5",
           "edicoes_anteriores/sorteio_34",
           "edicoes_anteriores/sorteio_35",
           "edicoes_anteriores/sorteio_36",
           "edicoes_anteriores/sorteio_37",
           "edicoes_anteriores/sorteio_38",
           "edicoes_anteriores/sorteio_39",
           "edicoes_anteriores/sorteio_40"]

seq_folders = []
file_names = []
file_names_and_paths = []

for folder in folders:
    directory = '../
       programa_de_fiscalizacao_em_entes_federativos/' +
       folder
```

```
    number_of_files = len([name for name in os.listdir(
       directory) if os.path.isfile(os.path.join(directory,
       name))])

    for i in range(0, number_of_files):
        file_name_and_path = directory + "/" + os.listdir(
           directory)[i]
        if (".pdf" in file_name_and_path):
            seq_folders.append(folder)
            file_names.append(os.listdir(directory)[i])
            file_names_and_paths.append(file_name_and_path)

print('Example: \n' + file_names_and_paths[0:1][0])
```

Generate target feature for each report ('read' and summarised the polarity)

```
sentilex_database = pd.read_csv("../sentilex/99
   _01_sentilex_database.csv",
                                sep = ";")

sentilex_database.adjective = sentilex_database.adjective.
   str.normalize('NFKD').\
                                str.encode('ascii', errors='
                                   ignore').str.decode('utf
                                   -8')
```

```
cities = pd.DataFrame()

print("List of reports read and summarised")

for file_number in range(0, len(file_names_and_paths)):
    folder = seq_folders[file_number]
    file_name = file_names[file_number]
    file_name_and_path = file_names_and_paths[file_number]
    print(str(datetime.datetime.now()) + ' ' +
       file_name_and_path)

    # read report using external library pdf miner and save
       in 'temp_report.txt'
    command_to_cmd = 'pdf2txt.py "' + file_name_and_path + '
       " > temp_report.txt'
    os.system(command_to_cmd)

    # read temporary file
    temporary_file = open('temp_report.txt', 'r')
```

```
whole_text = ''

for line in temporary_file:
    whole_text += line

words = re.findall(r"[\w']+", unidecode.unidecode(re.sub
   ('\d', ' ', whole_text).lower()))

# create the frequencies
words_freq = pd.DataFrame.from_dict(Counter(words),
   orient = 'index').reset_index()
words_freq.columns = ['word', 'freq']
words_freq['pct'] = words_freq['freq']/sum(words_freq.
   freq)

# aggregate polarity
words_freq_polarity = words_freq.merge(sentilex_database
   ,
                                       left_on = "word",
                                       right_on = "
                                          adjective",
                                       how = "left").
                                          iloc[:, [0, 1,
                                           2, 4]]

# summarise
number_of_words = words_freq_polarity.freq.sum()
pct_pol_neg = words_freq_polarity[words_freq_polarity.
   polarity == -1].pct.sum()
pct_pol_pos = words_freq_polarity[words_freq_polarity.
   polarity == 1].pct.sum()
pct_pol_neu = words_freq_polarity[words_freq_polarity.
   polarity == 0].pct.sum()
pct_pol_missing = words_freq_polarity[
   words_freq_polarity.polarity.isna()].pct.sum()

current_city = pd.DataFrame({"folder": folder,
                             "file_name": file_name,
                             "number_of_words":
                                number_of_words,
                             "pct_pol_neg": pct_pol_neg,
                             "pct_pol_pos": pct_pol_pos,
                             "pct_pol_neu": pct_pol_neu,
                             "pct_pol_missing":
                                pct_pol_missing},
                            index = [0])

cities = cities.append(current_city)
```

```
    # save last words_freq_polarity dataframe as an example
    if file_number + 1 == len(file_names_and_paths):
        words_freq_polarity.to_csv('temp_words_freq_polarity
           .csv',
                                   sep=';',
                                   encoding='utf-8',
                                   index=False)
```

```
len(file_names_and_paths)
```

```
os.remove("temp_report.txt")
```

```
cities.to_csv("../target_feature/01_target_feature.csv",
              sep=';',
              encoding='utf-8',
              index=False)
```

A.7 Data Processing (raw dataset)

```
import pandas as pd
import os
```

Read and handle target feature

```
target_feature = pd.read_csv('../target_feature/01
   _target_feature.csv',
                             sep=';')
```

```
target_feature.head()
```

```
target_feature['temp'] = target_feature['file_name'].str.
   replace('[0-9]|.pdf|-', ' ', regex=True)\
     .str.normalize('NFKD').str.encode('ascii', errors='
        ignore').str.decode('utf-8').str.lower().str.strip()
target_feature['city'] = target_feature['temp'].str[:-3]
target_feature['state'] = target_feature['temp'].str[-2:]
target_feature['city_state'] = target_feature['city'].map(
   str) + '_' + target_feature['state']
```

```
target_feature = target_feature.drop("temp", axis=1)
target_feature.head()
```

Read explanatory features: education, family, fertility and work (2000 and 2010)

```
raw_dataset = target_feature
```

```
state_name_to_acronym = pd.DataFrame({'full_state_name':
                                      ['acre',
                                       'alagoas',
                                       'amapa',
                                       'amazonas',
                                       'bahia',
                                       'ceara',
                                       'distrito_federal',
                                       'espirito_santo',
                                       'goias',
                                       'maranhao',
                                       'mato_grosso',
                                       'mato_grosso_do_sul',
                                       'minas_gerais',
                                       'para',
                                       'paraiba',
                                       'parana',
                                       'pernambuco',
                                       'piaui',
                                       'rio_de_janeiro',
                                       'rio_grande_do_norte'
                                         ,
                                       'rio_grande_do_sul',
                                       'rondonia',
                                       'roraima',
                                       'santa_catarina',
                                       'sao_paulo',
                                       'sergipe',
                                       'tocantins'],
                                      'acronym': ['ac',
                                                  'al',
                                                  'ap',
                                                  'am',
                                                  'ba',
                                                  'ce',
                                                  'df',
                                                  'es',
```

```
                                                        'go',
                                                        'ma',
                                                        'mt',
                                                        'ms',
                                                        'mg',
                                                        'pa',
                                                        'pb',
                                                        'pr',
                                                        'pe',
                                                        'pi',
                                                        'rj',
                                                        'rn',
                                                        'rs',
                                                        'ro',
                                                        'rr',
                                                        'sc',
                                                        'sp',
                                                        'se',
                                                        'to']})

var_list = ['var_01',
            'var_02',
            'var_03']
```

```
paths = ['../ibge_censo/2000/education',
         '../ibge_censo/2000/family',
         '../ibge_censo/2000/fertility',
         '../ibge_censo/2000/work',
         '../ibge_censo/2010/education',
         '../ibge_censo/2010/family',
         '../ibge_censo/2010/fertility',
         '../ibge_censo/2010/work']
```

```
for path in paths:

    for var_name in var_list:
        full_temp = pd.DataFrame()

        for state in os.listdir(path):
            if not state.startswith('.'):
                state_acronym = state_name_to_acronym.loc[
                    state_name_to_acronym.full_state_name ==
                        state]['acronym'].values[0]

                for filename in os.listdir(path + '/' +
                   state):
```

```
                    if not filename.startswith('.') and
                       filename.endswith(var_name + '.csv'):

                        temp = pd.read_csv(path + '/' +
                           state + '/' + filename)
                        temp['city_state'] = temp['city'].
                           map(str) + '_' + state_acronym

                        full_temp = pd.concat([full_temp,
                           temp])

        if full_temp.shape[0] != 0:
            full_temp = full_temp.add_prefix(path.split("/")
               [2] + '_' + path.split("/")[3] + '_' +
               var_name + '_')
            column_to_join = path.split("/")[2] + '_' + path
               .split("/")[3] + '_' + var_name + '
               _city_state'

            raw_dataset = pd.merge(raw_dataset,
                                   full_temp.iloc[:,1:],
                                   left_on="city_state",
                                   right_on=column_to_join,
                                   how="left")

            raw_dataset = raw_dataset.drop(column_to_join,
               axis=1)

            print(path + ' [' + var_name + '] ')
```

Read explanatory feature: social indicator (different pattern)

```
paths = ['../ibge_censo/2010/social_indicator']
```

Changing city name due to city being known by two different names

```
raw_dataset.loc[raw_dataset.file_name=='3238-Sa o Vale rio
    da Natividade-TO.pdf', 'city_state'] = 'sao valerio_to'
```

```
for path in paths:

    for var_name in var_list:
        full_temp = pd.DataFrame()
```

```
        for state in os.listdir(path):
            if not state.startswith('.'):
                state_acronym = state_name_to_acronym.loc[
                    state_name_to_acronym.full_state_name ==
                        state]['acronym'].values[0]

                for filename in os.listdir(path + '//' +
                   state):
                    if not filename.startswith('.') and
                       filename.endswith(var_name + '.csv'):

                        temp = pd.read_csv(path + '//' +
                           state + '//' + filename)
                        temp['city_state'] = temp['city'].
                           map(str) + '_' + state_acronym

                        full_temp = pd.concat([full_temp,
                           temp])

        if full_temp.shape[0] != 0:
            full_temp = full_temp.add_prefix(path.split("/")
               [3] + '_' + var_name + '_')
            column_to_join = path.split("/")[3] + '_' +
               var_name + '_city_state'

            raw_dataset = pd.merge(raw_dataset,
                                   full_temp.iloc[:,1:],
                                   left_on="city_state",
                                   right_on=column_to_join,
                                   how="left")

            raw_dataset = raw_dataset.drop(column_to_join,
               axis=1)

            print(path + ' [' + var_name + '] ')
```

Read explanatory feature: enem score (different pattern)

```
paths = ['../enem/2000/2000_enem_score_var_01.csv',
         '../enem/2010/2010_enem_score_var_01.csv']
```

```
for path in paths:
    temp = pd.read_csv(path, sep=';')
    temp = temp.iloc[1:,:]
    temp = temp.add_prefix(path.split("/")[2] + '
       _enem_var_01_')
```

```
    column_to_join = path.split("/")[2] + '
       _enem_var_01_city_state'

    raw_dataset = pd.merge(raw_dataset,
                           temp,
                           left_on="city_state",
                           right_on=column_to_join,
                           how="left")

    raw_dataset = raw_dataset.drop(column_to_join, axis=1)

raw_dataset.iloc[:, 88:95] = raw_dataset.iloc[:, 88:95].
   fillna(0)
```

```
for column in raw_dataset.columns:
    print(column)
```

```
raw_dataset.to_csv('02_01_raw_dataset.csv',
                   sep=';',
                   index=False)
```

A.8 Data Processing (modeling dataset)

```
import pandas as pd
import numpy as np
```

Feature engineering

```
full_dataset = pd.read_csv('02_01_raw_dataset.csv',
                           sep=';')
```

```
full_dataset.head()
```

```
full_dataset = full_dataset.replace('-', 0)

full_dataset.iloc[:, 10:] = full_dataset.iloc[:, 10:].apply(
   pd.to_numeric)
```

For all the features created using IBGE, divide them from the position in 2000 by the position in 2010, with the following observations: **if the feature is not a proportion, then divide the feature by the population size of the year (2000 or 2010)**

```
full_dataset['education_var_01_quantity_pct'] = (
   full_dataset['2000_education_var_01_quantity'] /
   full_dataset['2000_family_var_02_quantity']) / (
   full_dataset['2010_education_var_01_quantity'] /
   full_dataset['2010_family_var_02_quantity'])

full_dataset['family_var_01_suitable_pct'] = (full_dataset['
   2000_family_var_01_suitable'] / full_dataset['2000
   _family_var_01_total']) / (full_dataset['2010
   _family_var_01_suitable'] / full_dataset['2010
   _family_var_01_total'])
full_dataset['family_var_01_semi_suitable_pct'] = (
   full_dataset['2000_family_var_01_semi_suitable'] /
   full_dataset['2000_family_var_01_total']) / (full_dataset
   ['2010_family_var_01_semi_suitable'] / full_dataset['2010
   _family_var_01_total'])
full_dataset['family_var_01_inappropriate_pct'] = (
   full_dataset['2000_family_var_01_inappropriate'] /
   full_dataset['2000_family_var_01_total']) / (full_dataset
   ['2010_family_var_01_inappropriate'] / full_dataset['2010
   _family_var_01_total'])

full_dataset['fertility_var_01_has_children_pct'] = (
   full_dataset['2000_fertility_var_01_has_children'] /
   full_dataset['2000_fertility_var_01_total']) / (
   full_dataset['2010_fertility_var_01_has_children'] /
   full_dataset['2010_fertility_var_01_total'])
full_dataset['fertility_var_01_children_born_pct'] = (
   full_dataset['2000_fertility_var_01_children_born'] /
   full_dataset['2000_fertility_var_01_total']) / (
   full_dataset['2010_fertility_var_01_children_born'] /
   full_dataset['2010_fertility_var_01_total'])
full_dataset['fertility_var_01_children_borned_live_pct'] =
   (full_dataset['2000_fertility_var_01_children_borned_live
   '] / full_dataset['2000_fertility_var_01_total']) / (
   full_dataset['2010_fertility_var_01_children_borned_live'
   ] / full_dataset['2010_fertility_var_01_total'])
full_dataset['fertility_var_01_children_borned_dead_pct'] =
   (full_dataset['2000_fertility_var_01_children_borned_dead
   '] / full_dataset['2000_fertility_var_01_total']) / (
   full_dataset['2010_fertility_var_01_children_borned_dead'
   ] / full_dataset['2010_fertility_var_01_total'])

full_dataset['fertility_var_02_married_pct'] = (full_dataset
```

```
   ['2000_fertility_var_02_married'] / full_dataset['2000
   _fertility_var_02_total']) / (full_dataset['2010
   _fertility_var_02_married'] / full_dataset['2010
   _fertility_var_02_total'])
full_dataset['fertility_var_02_separated_pct'] = (
   full_dataset['2000_fertility_var_02_separated'] /
   full_dataset['2000_fertility_var_02_total']) / (
   full_dataset['2010_fertility_var_02_separated'] /
   full_dataset['2010_fertility_var_02_total'])
full_dataset['fertility_var_02_divorced_pct'] = (
   full_dataset['2000_fertility_var_02_divorced'] /
   full_dataset['2000_fertility_var_02_total']) / (
   full_dataset['2010_fertility_var_02_divorced'] /
   full_dataset['2010_fertility_var_02_total'])
full_dataset['fertility_var_02_widow_pct'] = (full_dataset['
   2000_fertility_var_02_widow'] / full_dataset['2000
   _fertility_var_02_total']) / (full_dataset['2010
   _fertility_var_02_widow'] / full_dataset['2010
   _fertility_var_02_total'])
full_dataset['fertility_var_02_single_pct'] = (full_dataset[
   '2000_fertility_var_02_single'] / full_dataset['2000
   _fertility_var_02_total']) / (full_dataset['2010
   _fertility_var_02_single'] / full_dataset['2010
   _fertility_var_02_total'])

full_dataset['fertility_var_03_total_pct'] = (full_dataset['
   2000_fertility_var_03_total'] / full_dataset['2000
   _family_var_02_quantity']) / (full_dataset['2010
   _fertility_var_03_total'] / full_dataset['2010
   _family_var_02_quantity'])

full_dataset['work_var_01_regular_pct'] = ((full_dataset['
   2000_work_var_01_domestic_regular'] + full_dataset['2000
   _work_var_01_other_regular'] + full_dataset['2000
   _work_var_01_military_and_gov']) / full_dataset['2000
   _work_var_01_total']) / ((full_dataset['2010
   _work_var_01_main_regular'] + full_dataset['2010
   _work_var_01_other_regular']) / full_dataset['2010
   _work_var_01_total'])
full_dataset['work_var_01_irregular_pct'] = ((full_dataset['
   2000_work_var_01_domestic_irregular'] + full_dataset['
   2000_work_var_01_other_irregular']) / full_dataset['2000
   _work_var_01_total']) / ((full_dataset['2010
   _work_var_01_main_irregular'] + full_dataset['2010
   _work_var_01_other_irregular']) / full_dataset['2010
   _work_var_01_total'])

full_dataset['social_indicator_var_01_15_to_24_years_pct'] =
```

```
    (full_dataset['
   social_indicator_var_01_2000_15_to_24_years'] /
   full_dataset['social_indicator_var_01_2010_15_to_24_years
   '])
full_dataset['social_indicator_var_01_25_to_59_years_pct'] =
    (full_dataset['
   social_indicator_var_01_2000_25_to_59_years'] /
   full_dataset['social_indicator_var_01_2010_25_to_59_years
   '])
full_dataset['social_indicator_var_01_60_to_more_years_pct']
    = (full_dataset['
   social_indicator_var_01_2000_60_to_more_years'] /
   full_dataset['
   social_indicator_var_01_2010_60_to_more_years'])

full_dataset['social_indicator_var_02_suitable_pct'] =
   full_dataset['social_indicator_var_02_2000_suitable'] /
   full_dataset['social_indicator_var_02_2010_suitable']
full_dataset['social_indicator_var_02_semi_suitable_pct'] =
   full_dataset['social_indicator_var_02_2000_semi_suitable'
   ] / full_dataset['
   social_indicator_var_02_2010_semi_suitable']
full_dataset['social_indicator_var_02_inappropriate_pct'] =
   full_dataset['social_indicator_var_02_2000_inappropriate'
   ] / full_dataset['
   social_indicator_var_02_2010_inappropriate']

full_dataset['
   social_indicator_var_03_responsable_illiterate_pct'] =
   full_dataset['
   social_indicator_var_03_2000_responsable_illiterate'] /
   full_dataset['
   social_indicator_var_03_2010_responsable_illiterate']
full_dataset['
   social_indicator_var_03_inappropriate_residence_pct'] =
   full_dataset['
   social_indicator_var_03_2000_inappropriate_residence'] /
   full_dataset['
   social_indicator_var_03_2010_inappropriate_residence']
full_dataset['
   social_indicator_var_03_responsable_illiterate_and_inapprop
    riate_residence_pct'] = full_dataset['
   social_indicator_var_03_2000_responsable_illiterate_and_ina
    ppropriate_residence'] / full_dataset['
   social_indicator_var_03_2010_responsable_illiterate_and_ina

ppropriate_residence']
```

```
full_dataset['enem_var_01_enem_score_mean_pct'] =
   full_dataset['2000_enem_var_01_enem_score_mean'] /
   full_dataset['2010_enem_var_01_enem_score_mean']
full_dataset['enem_var_01_enem_score_std_pct'] =
   full_dataset['2000_enem_var_01_enem_score_std'] /
   full_dataset['2010_enem_var_01_enem_score_std']
full_dataset['enem_var_01_enem_score_median_pct'] =
   full_dataset['2000_enem_var_01_enem_score_median'] /
   full_dataset['2010_enem_var_01_enem_score_median']
```

For the target feature, change the proportion getting the relative **pct_pol_neg** over the sum of **pct_pol_neg** and **pct_pol_pos**

```
full_dataset.insert(0, 'pct_pol_neg_rel', full_dataset['
   pct_pol_neg']/(full_dataset['pct_pol_neg']+full_dataset['
   pct_pol_pos']))
```

Select only generated features and remove rows with NaN values on it

```
modeling_dataset = full_dataset.iloc[:, np.r_[0, 9,
   96:124]].dropna()
```

Remove 'inf' value for all columns

```
def replace_inf_by_max(df, col_name):
    max_for_column = max(df.loc[df[col_name] != np.inf,
       col_name])
    df.loc[df[col_name] == np.inf, col_name] =
       max_for_column

    return df
```

```
for col in modeling_dataset.columns:
    if col not in ['enem_var_01_enem_score_mean_pct', '
       enem_var_01_enem_score_std_pct', '
       enem_var_01_enem_score_median_pct']:
        modeling_dataset = replace_inf_by_max(
           modeling_dataset, col)
```

One hot encoding for 'state' feature

```
modeling_dataset = pd.concat([
    modeling_dataset,
    pd.get_dummies(modeling_dataset['state'], prefix = '
       state')
], axis = 1)

modeling_dataset = modeling_dataset.drop(columns=['state'])
```

```
modeling_dataset.tail()
```

```
for col in modeling_dataset.columns:
    print(col)
```

```
modeling_dataset.to_csv('02_02_modeling_dataset.csv',
                        sep=';',
                        index=False)
```

A.9 Data Processing (training and validation dataset)

```
import pandas as pd
```

Splitting the datasets (training and validation)

```
modeling_dataset = pd.read_csv('02_02_modeling_dataset.csv',
                               sep=';')
```

```
training_dataset = modeling_dataset.sample(frac=0.75,
                                           random_state=7)
validation_dataset = modeling_dataset.drop(training_dataset.
   index)
```

```
print('=== Number of rows === \n' +
      'Training: ' + str(len(training_dataset)) + '\n' +
      'Validation: ' + str(len(validation_dataset)))
```

```
training_dataset.head()
```

```
training_dataset.to_csv('02_03_training_dataset.csv',
                        sep=';',
                        index=False)
```

```
validation_dataset.to_csv('02_03_validation_dataset.csv',
                          sep=';',
                          index=False)
```

A.10 Model Visualization

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
training_dataset = pd.read_csv('../2_database/
   data_processing/02_03_training_dataset.csv',
                                sep=';')

validation_dataset = pd.read_csv('../2_database/
   data_processing/02_03_validation_dataset.csv',
                                  sep=';')
```

```
df = pd.concat([training_dataset, validation_dataset])
```

```
df.head()
```

```
for col in df.columns:
    print(col)
```

```
for i in range(0, len(df.columns)):
    plt.scatter(df.iloc[:, 0], df.iloc[:, i])

    plt.title(df.columns[i])
    plt.xlabel(df.columns[0])

    plt.savefig("img/two_by_two_scatter_plot/" +
                str(i).rjust(2, '0') + "_" + df.columns[i] +
                    ".png")
    plt.show()
```

A.11 Model Training

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import numpy as np
import re
import datetime
from numpy import inf
from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_randFloat
import pickle
import shap
import warnings
warnings.simplefilter('ignore')

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn import model_selection
from sklearn.model_selection import RandomizedSearchCV

from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
```

```
training_dataset = pd.read_csv('../2_database/
   data_processing/02_03_training_dataset.csv',
                                sep=';')

validation_dataset = pd.read_csv('../2_database/
   data_processing/02_03_validation_dataset.csv',
                                  sep=';')
```

```
print("Training and Validation size: " + str(
   training_dataset.shape) + " / " + str(validation_dataset.
   shape))
```

```
Training and Validation size: (420, 53) / (140, 53)
```

```
training_dataset.head()
```

```
for col in training_dataset.columns:
    print(col)
```

```
array = training_dataset.values

X_training = array[:, 1:]
Y_training = array[:, 0]
```

```
array = validation_dataset.values

X_validation = array[:, 1:]
Y_validation = array[:, 0]
```

Function for regression evaluation

```
def regression_evaluation(Y_training, y_training_pred,
   Y_validation, y_validation_pred):

    random_simulation = np.random.randint(int(np.quantile(
       Y_training, 0)*10), int(np.quantile(Y_training, 1)
       *10), size=len(Y_validation)) * 0.1
    null_simulation = [np.mean(Y_training)] * len(
       Y_validation)

    print(
        "RMSE training: " + str(np.round(np.sqrt(
           mean_squared_error(Y_training, y_training_pred)),
            4)) + "\n" +
        "RMSE validation: " + str(np.round(np.sqrt(
           mean_squared_error(Y_validation,
           y_validation_pred)), 4)) + "\n" +
        "RMSE validation random model: " + str(np.round(np.
           sqrt(mean_squared_error(Y_validation,
           random_simulation)), 4)) + "\n" +
        "RMSE validation null model: " + str(np.round(np.
           sqrt(mean_squared_error(Y_validation,
           null_simulation)), 4))
    )

    print("\n")

    delta = 0.15
```

```
overestimate_training_rate = np.round(sum((
   y_training_pred > Y_training * (1 + delta)) == True)/
   len(Y_training), 4)
underestimate_training_rate = np.round(sum((
   y_training_pred < Y_training * (1 - delta)) == True)/
   len(Y_training), 4)
wellestimate_training_rate = np.round(1-(
   overestimate_training_rate +
   underestimate_training_rate), 4)

overestimate_validation_rate = np.round(sum((
   y_validation_pred > Y_validation * (1 + delta)) ==
   True)/len(Y_validation), 4)
underestimate_validation_rate = np.round(sum((
   y_validation_pred < Y_validation * (1 - delta)) ==
   True)/len(Y_validation), 4)
wellestimate_validation_rate = np.round(1-(
   overestimate_validation_rate +
   underestimate_validation_rate), 4)

overestimate_random_rate = np.round(sum((
   random_simulation > Y_validation * (1 + delta)) ==
   True)/len(Y_validation), 4)
underestimate_random_rate = np.round(sum((
   random_simulation < Y_validation * (1 - delta)) ==
   True)/len(Y_validation), 4)
wellestimate_random_rate = np.round(1-(
   overestimate_random_rate + underestimate_random_rate)
   , 4)

overestimate_null_rate = np.round(sum((null_simulation >
    Y_validation * (1 + delta)) == True)/len(
   Y_validation), 4)
underestimate_null_rate = np.round(sum((null_simulation
   < Y_validation * (1 - delta)) == True)/len(
   Y_validation), 4)
wellestimate_null_rate = np.round(1-(
   overestimate_null_rate + underestimate_null_rate), 4)

print(
    "BANDS training (underestimate | well | overestimate
       ): " + str(underestimate_training_rate) + " | " +
        str(wellestimate_training_rate) + " | " + str(
       overestimate_training_rate) + "\n" +
    "BANDS validation (underestimate | well |
       overestimate): " + str(
       underestimate_validation_rate) + " | " + str(
       wellestimate_validation_rate) + " | " + str(
```

```
            overestimate_validation_rate) + "\n" +
        "BANDS validation random model (underestimate | well
            | overestimate): " + str(
            underestimate_random_rate) + " | " + str(
            wellestimate_random_rate) + " | " + str(
            overestimate_random_rate) + "\n" +
        "BANDS validation null model (underestimate | well |
            overestimate): " + str(underestimate_null_rate)
            + " | " + str(wellestimate_null_rate) + " | " +
            str(overestimate_null_rate)
    )

    figure(num=None, figsize=(8, 6), dpi=80, facecolor='w',
        edgecolor='k')

    plt.plot([0, 1], [0, 1], 'k-', zorder=1)
    plt.scatter(Y_training, y_training_pred, label='training
        ', zorder=2)
    plt.scatter(Y_validation, y_validation_pred, label='
        validation', zorder=3)

    plt.ylabel("predicted")
    plt.xlabel("observed")
    plt.xlim([0, 0.75])
    plt.ylim([0, 0.75])

    plt.legend()
```

Function for printing shapley value importance

```
def print_shapley_value_importance(model, X_matrix,
    feature_names):
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(X_matrix)

    shap.summary_plot(shap_values,
                      X_matrix,
                      feature_names=feature_names)
```

1 - Linear Regression

```
linear_regression_model = LinearRegression()
linear_regression_model.fit(X_training, Y_training)
```

```
y_training_pred = linear_regression_model.predict(X_training
   )
y_validation_pred = linear_regression_model.predict(
   X_validation)
```

```
linear_regression_model = pickle.load(open('03
   _02_linear_regression_model.pickle', 'rb'))

y_training_pred = linear_regression_model.predict(X_training
   )
y_validation_pred = linear_regression_model.predict(
   X_validation)
```

```
regression_evaluation(Y_training,
                      y_training_pred,
                      Y_validation,
                      y_validation_pred)
```

```
pickle.dump(linear_regression_model, open('03
   _02_linear_regression_model.pickle', 'wb'))
```

```
linear_regression_model = pickle.load(open('03
   _02_linear_regression_model.pickle', 'rb'))
```

```
i=0

for column_name in training_dataset.columns[1:]:
    print(column_name + " : " + str(linear_regression_model.
       coef_[i]))
    i = i+1
```

2 - Random Forest

```
random_forest_model = RandomForestRegressor(random_state=7)

params = {'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
          'max_features': sp_randint(5, len(training_dataset
             .columns)-1),
          'min_samples_split': sp_randFloat(0.1, 0.8),
          'min_samples_leaf': sp_randFloat(0.05, 0.4),
          'n_estimators': [10, 25, 50, 75, 100],
```

```
          'bootstrap': [True, False],
          'criterion': ['mse']}

rs_random_forest_model = RandomizedSearchCV(
    random_forest_model,
    param_distributions=params,
    n_iter=300,
    cv=5,
    iid=False,
    refit=True,
    random_state=7)

rs_random_forest_model.fit(X_training, Y_training)

y_training_pred = rs_random_forest_model.predict(X_training)
y_validation_pred = rs_random_forest_model.predict(
   X_validation)
```

```
rs_random_forest_model = pickle.load(open('03
   _02_rs_random_forest_model.pickle', 'rb'))

y_training_pred = rs_random_forest_model.predict(X_training)
y_validation_pred = rs_random_forest_model.predict(
   X_validation)
```

```
regression_evaluation(Y_training,
                      y_training_pred,
                      Y_validation,
                      y_validation_pred)
```

```
pickle.dump(rs_random_forest_model, open('03
   _02_rs_random_forest_model.pickle', 'wb'))
```

```
rs_random_forest_model = pickle.load(open('03
   _02_rs_random_forest_model.pickle', 'rb'))
```

```
rs_random_forest_model.best_params_
```

```
random_forest_model = RandomForestRegressor(
    bootstrap=False,
    criterion='mse',
    max_depth=10,
```

```
    max_features=13,
    min_samples_leaf=0.05170415106602393,
    min_samples_split=0.4774005510811298,
    n_estimators=75,
    random_state=7)

random_forest_model.fit(X_training, Y_training)
```

```
print_shapley_value_importance(random_forest_model,
                               X_validation,
                               training_dataset.columns[1:])
```

3 - XGBoost

```
xgboost_model = XGBRegressor(random_state=7)

params = {'silent': [False],
          'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
          'learning_rate': [0.001, 0.01, 0.1, 1],
          'colsample_bytree': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
              1.0],
          'colsample_bylevel': [0.4, 0.5, 0.6, 0.7, 0.8,
             0.9, 1.0],
          'min_child_weight': [0.5, 1.0, 3.0, 5.0, 7.0,
             10.0],
          'gamma': [0, 0.25, 0.5, 1.0],
          'reg_lambda': [0.1, 1.0, 5.0, 10.0, 50.0, 100.0],
          'n_estimators': [10, 25, 50, 75, 100],
          'eval_metric': ['rmse']}

rs_xgboost_model = RandomizedSearchCV(xgboost_model,
                                      param_distributions=
                                         params,
                                      n_iter=300,
                                      cv=5,
                                      iid=False,
                                      refit=True,
                                      random_state=7)

rs_xgboost_model.fit(X_training, Y_training)

y_training_pred = rs_xgboost_model.predict(X_training)
y_validation_pred = rs_xgboost_model.predict(X_validation)
```

```
rs_xgboost_model = pickle.load(open('03_02_rs_xgboost_model.
   pickle', 'rb'))

y_training_pred = rs_xgboost_model.predict(X_training)
y_validation_pred = rs_xgboost_model.predict(X_validation)
```

```
regression_evaluation(Y_training,
                      y_training_pred,
                      Y_validation,
                      y_validation_pred)
```

```
pickle.dump(rs_xgboost_model, open('03_02_rs_xgboost_model.
   pickle', 'wb'))
```

```
rs_xgboost_model = pickle.load(open('03_02_rs_xgboost_model.
   pickle', 'rb'))
```

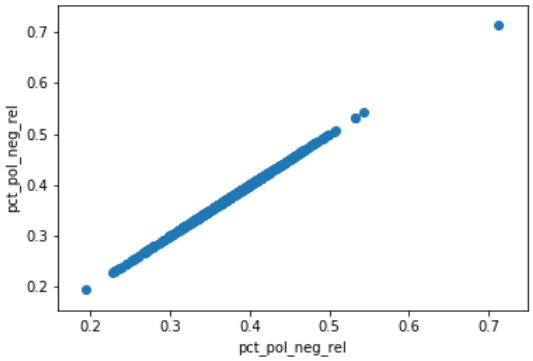
```
rs_xgboost_model.best_params_
```

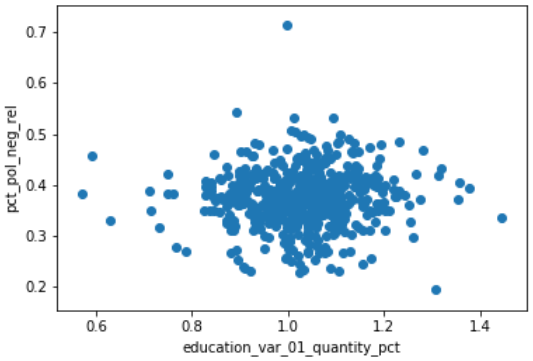
```
xgboost_model = XGBRegressor(silent=False,
                             reg_lambda=100.0,
                             n_estimators=50,
                             min_child_weight=7.0,
                             max_depth=12,
                             learning_rate=0.1,
                             gamma=0,
                             eval_metric='rmse',
                             colsample_bytree=0.4,
                             colsample_bylevel=0.7,
                             random_state=7)

xgboost_model.fit(X_training, Y_training)
```

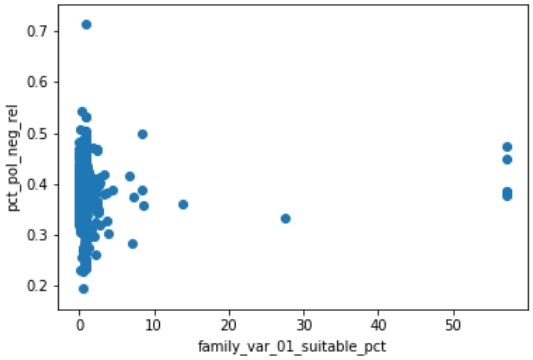
```
print_shapley_value_importance(xgboost_model,
                               X_validation,
                               training_dataset.columns[1:])
```

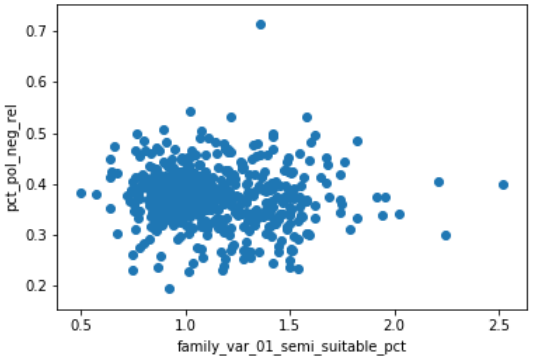
Apêndice B

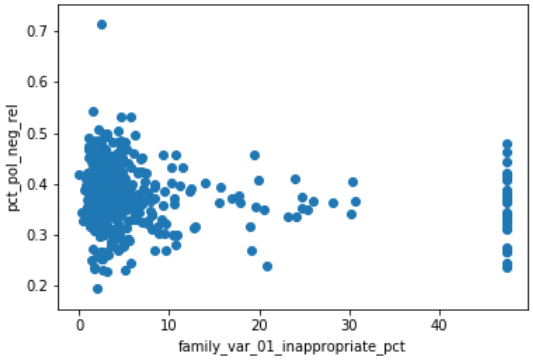
Gráficos

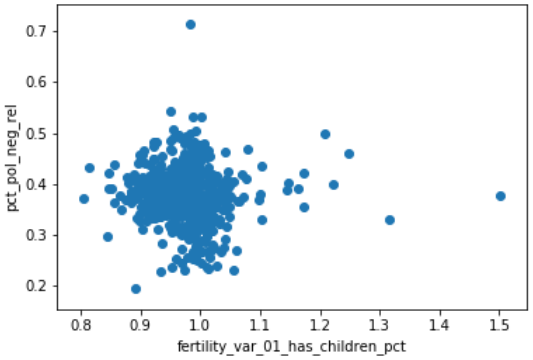


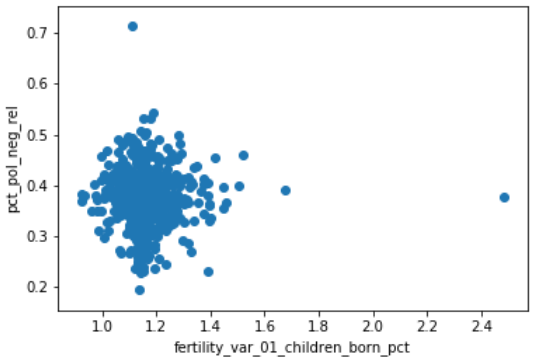


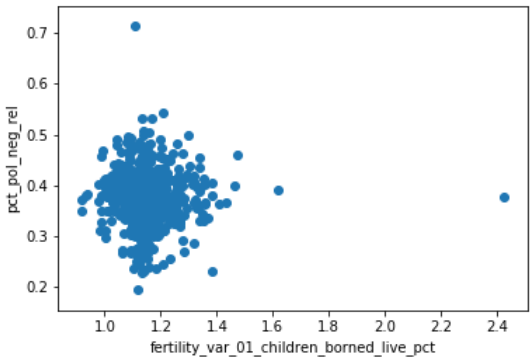


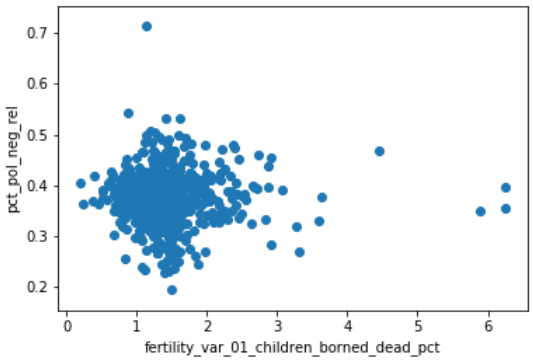


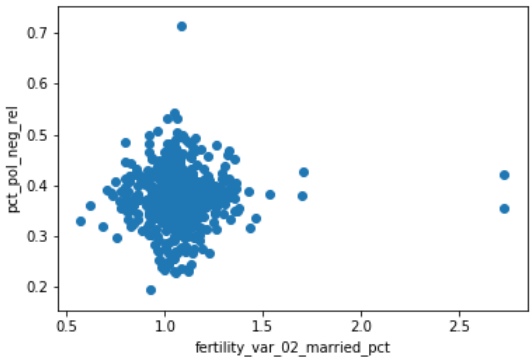


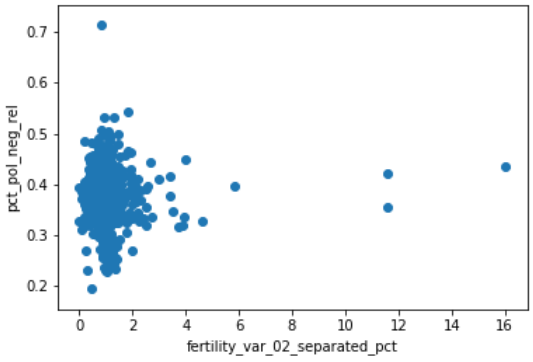


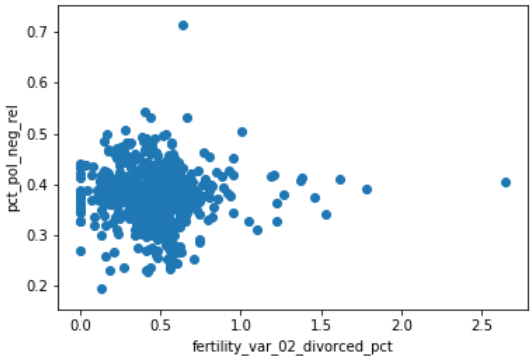

pct_pol_neg_rel
fertility_var_01_children_born_pct

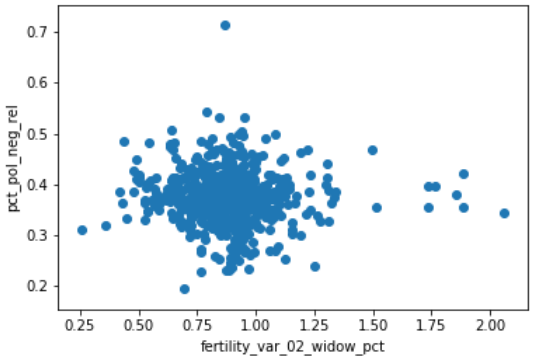

pct_pol_neg_rel
fertility_var_01_children_borned_live_pct

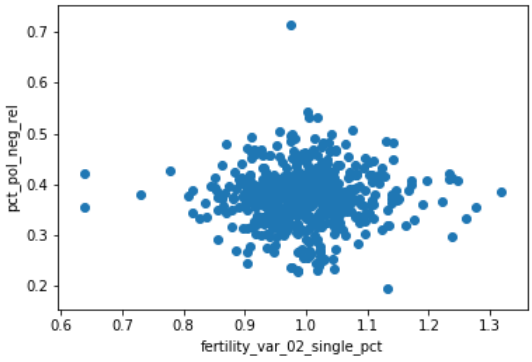

pct_pol_neg_rel
fertility_var_01_children_borned_dead_pct

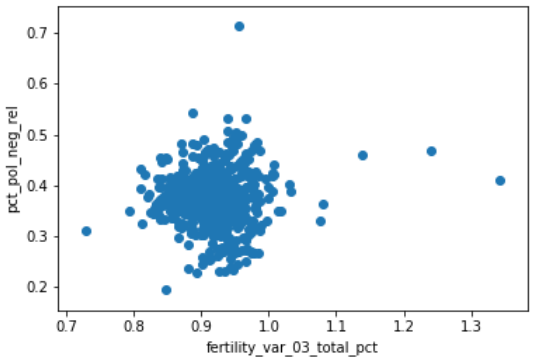

pct_pol_neg_rel
fertility_var_02_married_pct

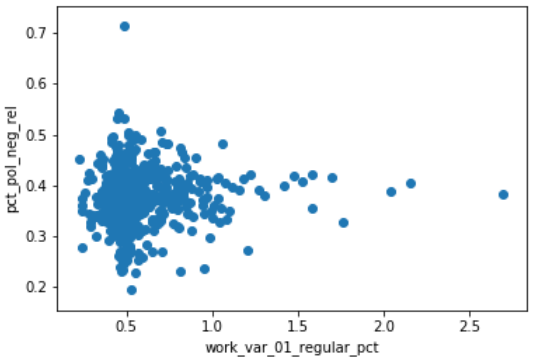

pct_pol_neg_rel
fertility_var_02_separated_pct

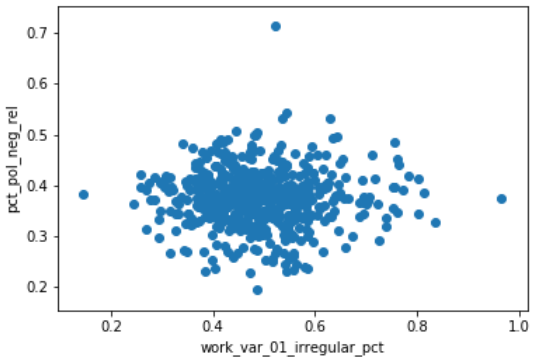

pct_pol_neg_rel
fertility_var_02_divorced_pct

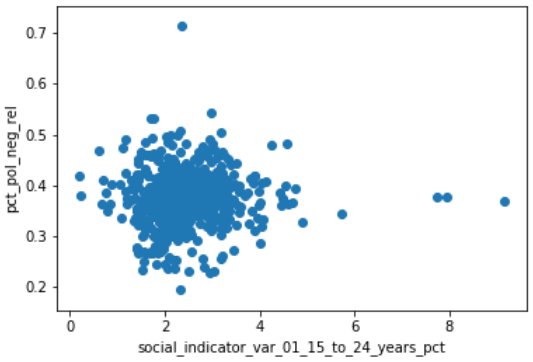

pct_pol_neg_rel
fertility_var_02_widow_pct

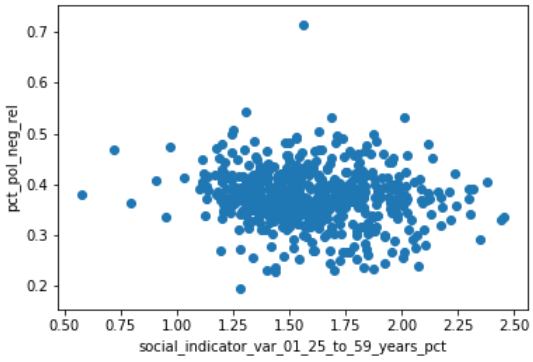

pct_pol_neg_rel
fertility_var_02_single_pct

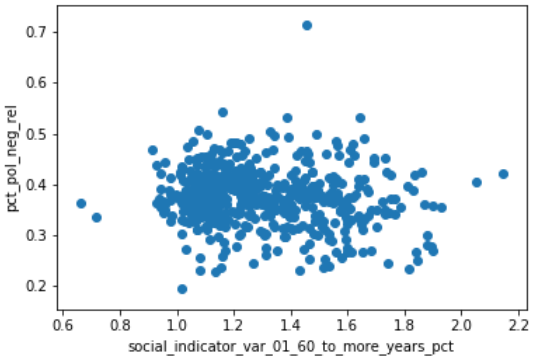

pct_pol_neg_rel
fertility_var_03_total_pct

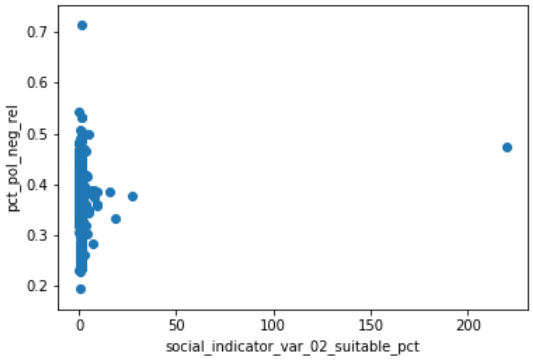

pct_pol_neg_rel
work_var_01_regular_pct

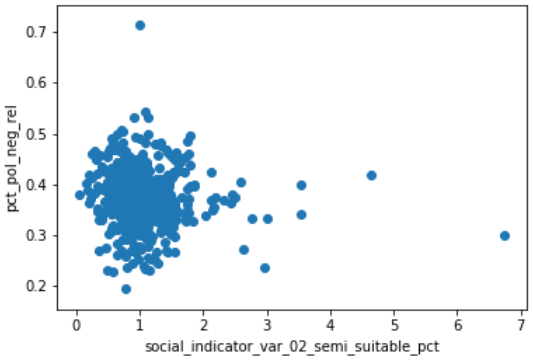

pct_pol_neg_rel
work_var_01_irregular_pct

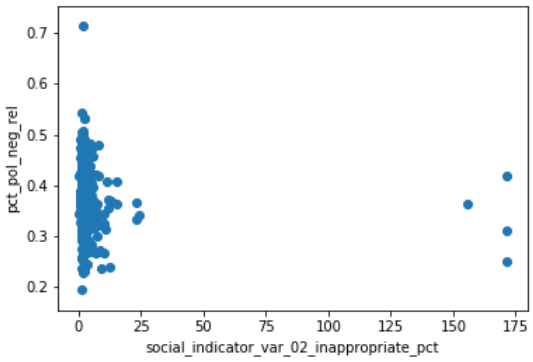

pct_pol_neg_rel
social_indicator_var_01_15_to_24_years_pct

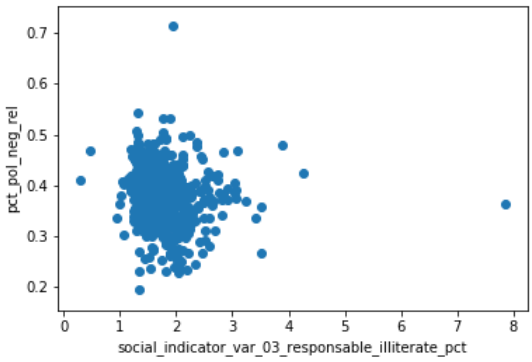

pct_pol_neg_rel
social_indicator_var_01_25_to_59_years_pct

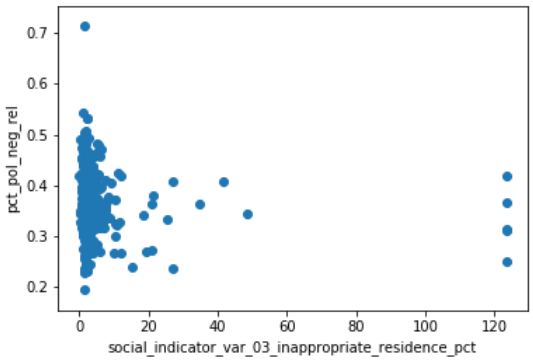

pct_pol_neg_rel
social_indicator_var_01_60_to_more_years_pct

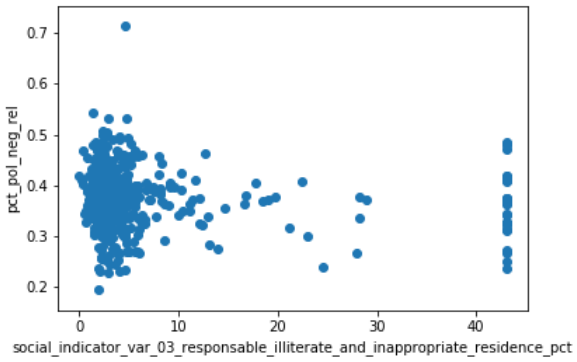

pct_pol_neg_rel
social_indicator_var_02_suitable_pct

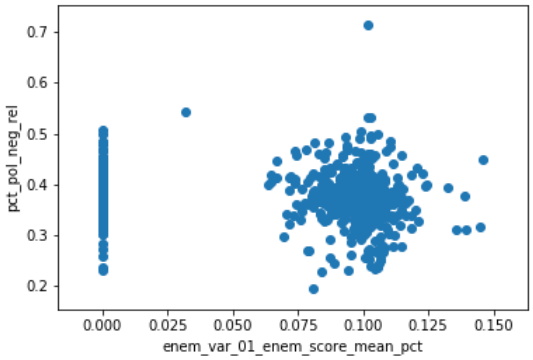

pct_pol_neg_rel
social_indicator_var_02_semi_suitable_pct

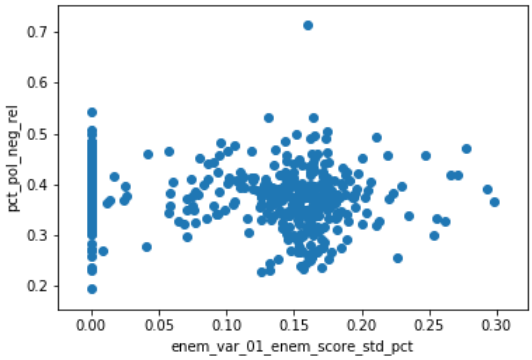

pct_pol_neg_rel
social_indicator_var_02_inappropriate_pct

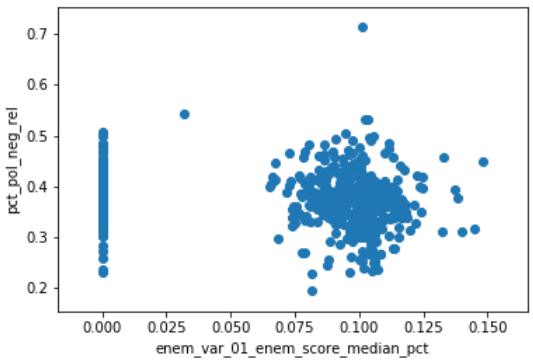

pct_pol_neg_rel
social_indicator_var_03_responsable_illiterate_pct

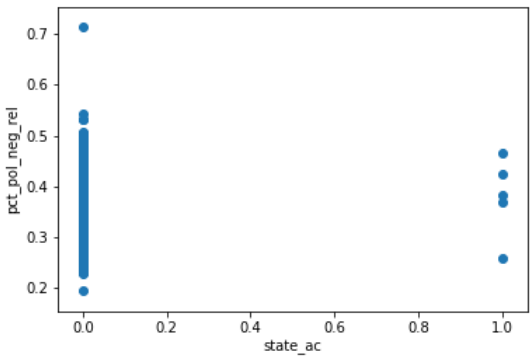

pct_pol_neg_rel
social_indicator_var_03_inappropriate_residence_pct

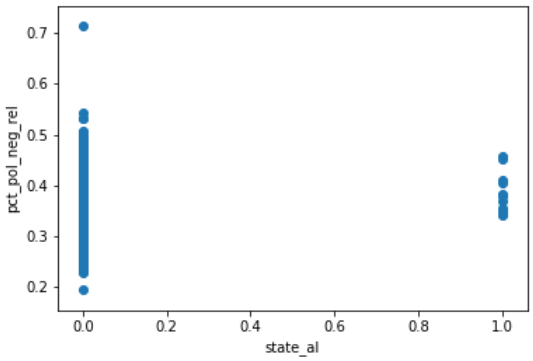

pct_pol_neg_rel
social_indicator_var_03_responsable_illiterate_and_inappropriate_residence_pct

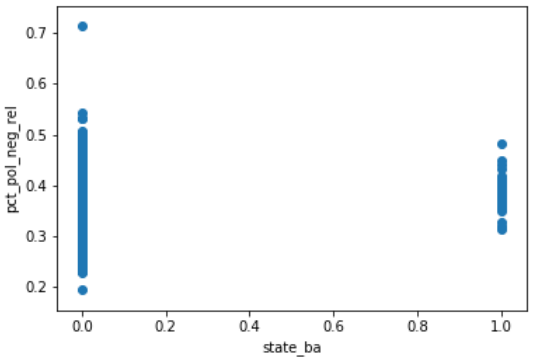

pct_pol_neg_rel
enem_var_01_enem_score_mean_pct

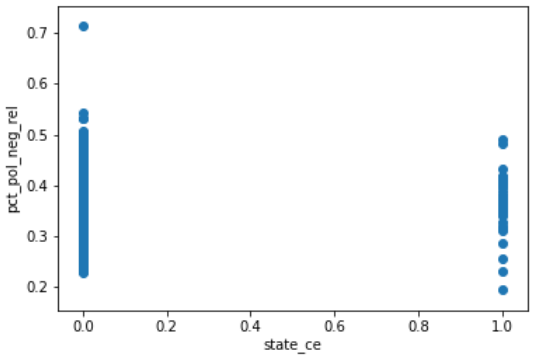

pct_pol_neg_rel
enem_var_01_enem_score_std_pct

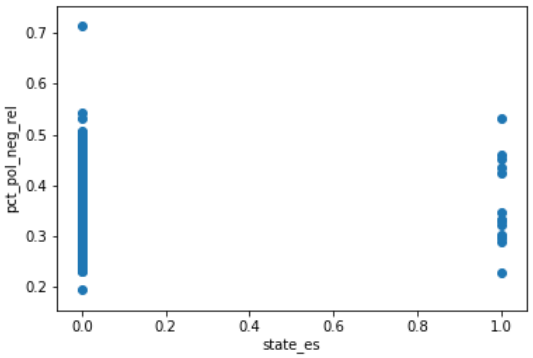

pct_pol_neg_rel
enem_var_01_enem_score_median_pct

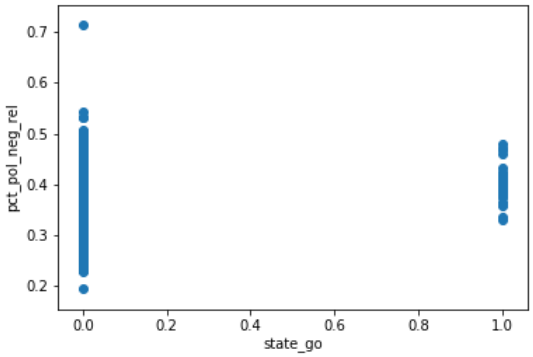

pct_pol_neg_rel
state_ac

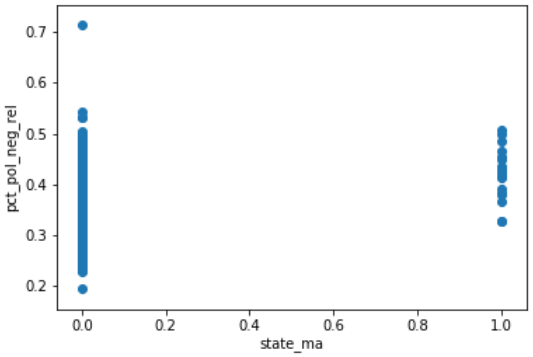

pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_al

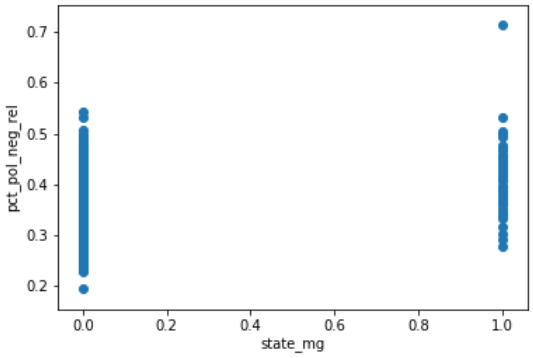

pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_ba

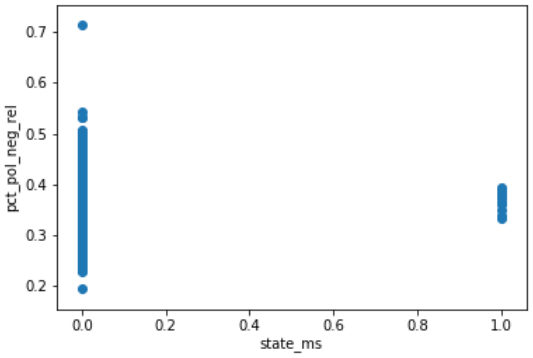

pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_ce

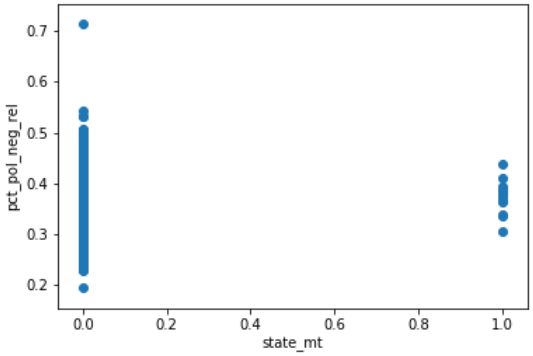

pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_es

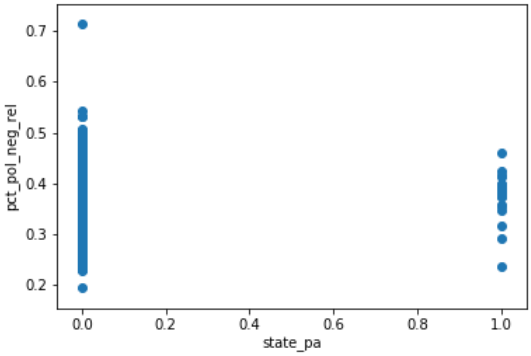

pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_go

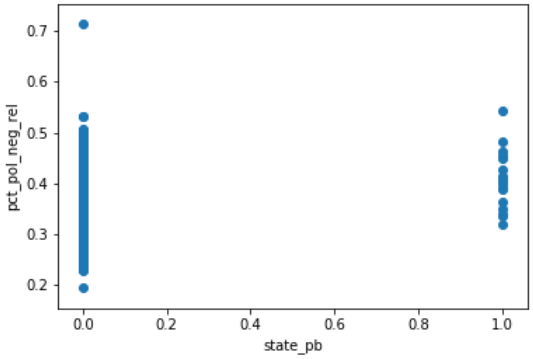

pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_ma

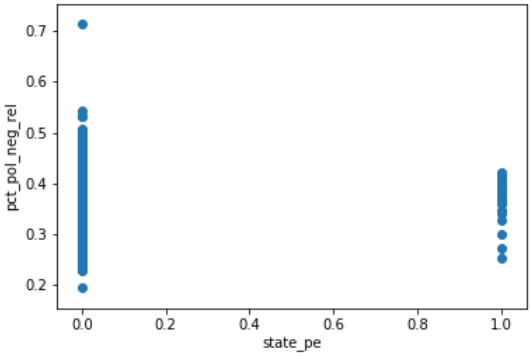

pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_mg

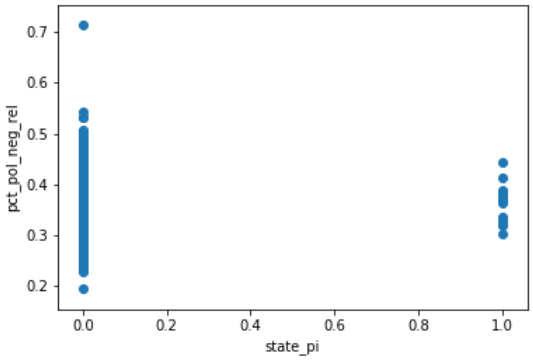

pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_ms

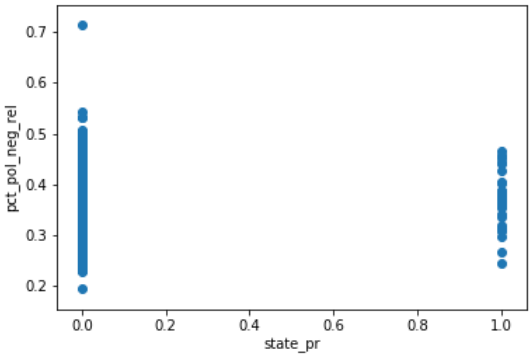

pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_mt

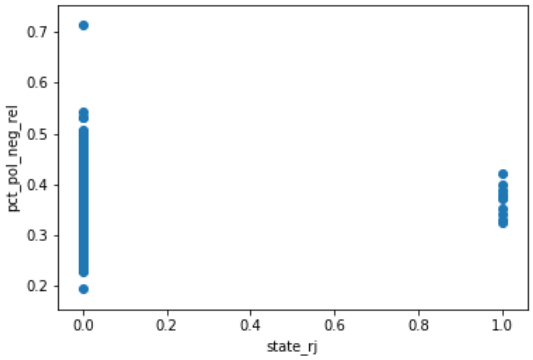

pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_pa

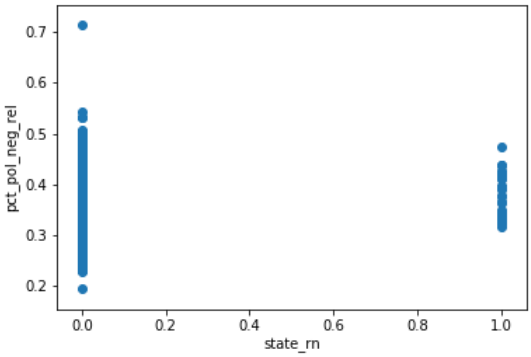

pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_pb


pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_pe


pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_pi


pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_pr


pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_rj


pct_pol_neg_rel
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
state_rn

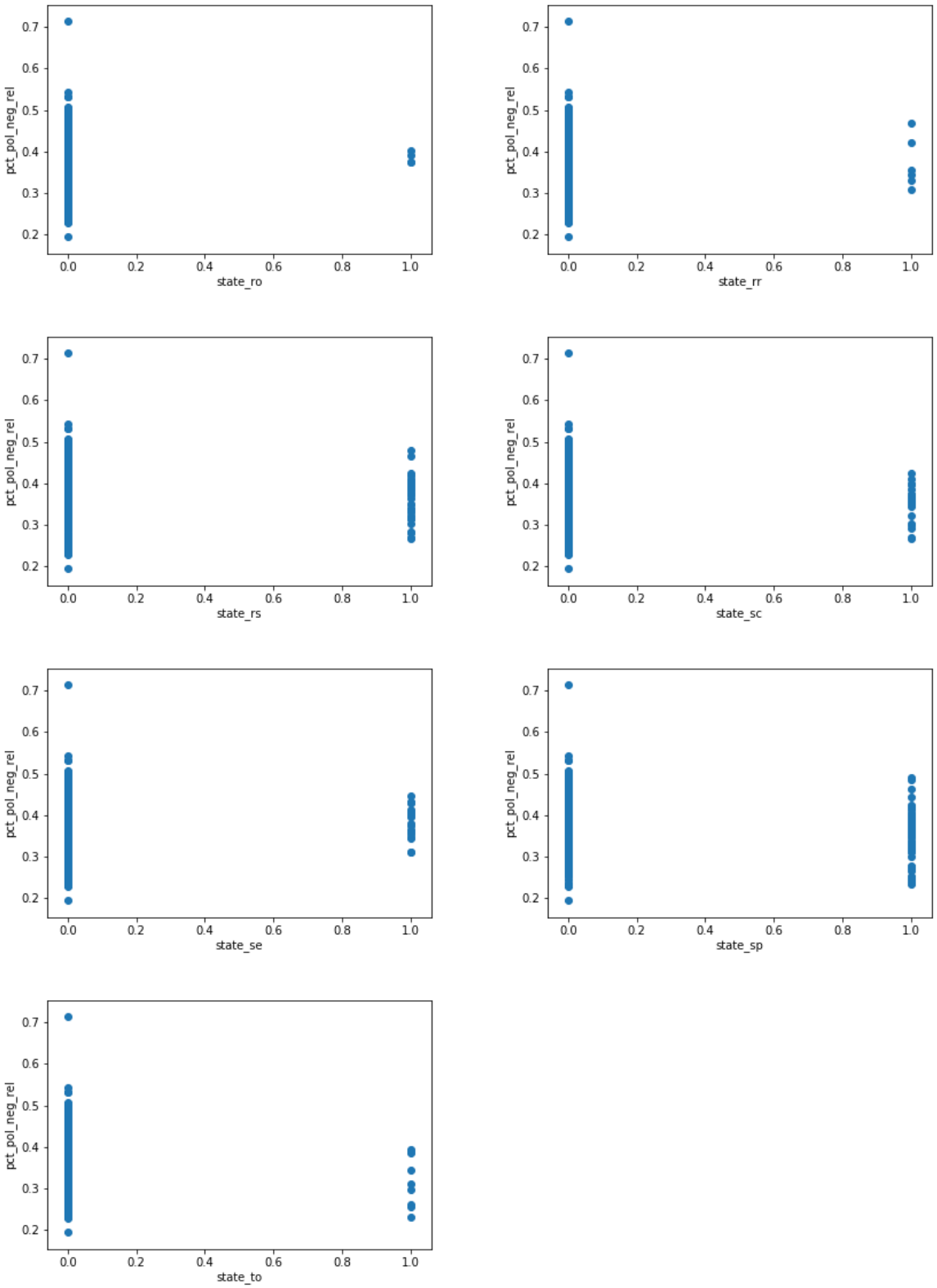


Figura B.1: *Gráfico de dispersão das variáveis explicativas e a variável resposta*

Bibliografia

Alves(2018) V. Alves. A relacao entre a desigualdade e a corrupcao politica na filosofia de rousseau. *Revista de Filosofia, Amargosa - BA*, 17:85–96. Citado na pág. 1

Becker e Tumitan(2014) K. Becker e D. Tumitan. Sentiment-based features for predicting election polls: a case study on the brazilian scenario. *IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, 2:126–133. Citado na pág. 4

Bergstra e Bengio(2012) J. Bergstra e Y. Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research 13*, páginas 281–305. Citado na pág. 14

Breiman(1997) L. Breiman. Arcing the edge. *Technical Report 486. Statistics Department, University of California, Berkeley.* Citado na pág. 26

Bruscato(2018) L. Bruscato. Repositório no GitHub pertencente ao Lucas Bruscato. https://github.com/lucasbruscato/Master, 2018. [Online; accessed 27-June-2019]. Citado na pág. 3

CGU(2015) CGU. Programa de Fiscalização em Entes Federativos. https://www.cgu.gov.br/assuntos/auditoria-e-fiscalizacao/programa-de-fiscalizacao-em-entes-federativos, 2015. [Online; accessed 27-June-2019]. Citado na pág. 1

Chen(2014) T. Chen. Xgboost. https://xgboost.readthedocs.io/, 2014. [Online; accessed 27-September-2019]. Citado na pág. 27

Cournapeau(2007) D. Cournapeau. Scikit-learn. https://scikit-learn.org/, 2007. [Online; accessed 12-September-2019]. Citado na pág. 14, 18, 22

ESPRIT(1996) ESPRIT. Cross-industry standard process for data mining. https://www.sv-europe.com/crisp-dm-methodology/, 1996. [Online; accessed 27-June-2019]. Citado na pág. 3

Ferraz e Finan(2008) C. Ferraz e F. Finan. Exposing corrupt politicians: the effects of brazil's publicly released audits on electoral outcomes. *The Quarterly Journal of Economics*, 1:703–745. Citado na pág. 1, 2, 4

Freedman(2009) D. Freedman. *Statistical Models: Theory and Practice.* Cambridge University Press. Citado na pág. 18

Friedman(1999a) J. Friedman. Greedy function approximation: A gradient boosting machine. *IMS 1999 Reitz Lecture.* Citado na pág. 26

Friedman(1999b) J. Friedman. Stochastic gradient boosting. Citado na pág. 26

Goldani(2001) MZ. et al. Goldani. Socio-economic status and infant mortality. *Revista Saude Publica*, 35:256–261. Citado na pág. 2

Hastie e Friedman(2001) Tibshirani R. Hastie, T. e J. Friedman. *The Elements of Statistical Learning*. Springer. Citado na pág. 26

Ho(1995) T. Ho. Random decision forests. *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, página 278–282. Citado na pág. 21

Huggins(2004) J. Huggins. Selenium framework. https://www.seleniumhq.org/projects/webdriver/, 2004. [Online; accessed 27-June-2019]. Citado na pág. 4

IBGE(2000) IBGE. Censo demográfico do IBGE. https://www.ibge.gov.br/estatisticas-novoportal/sociais/saude/9662-censo-demografico-2010.html?edicao=14881&t=downloads, 2000. [Online; accessed 27-June-2019]. Citado na pág. 9, 10

Kleinberg(1990) E. Kleinberg. Stochastic discrimination. *Annals of Mathematics and Artificial Intelligence*, página 207–239. Citado na pág. 21

Kohavi(1995) R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. *International joint Conference on artificial intelligence*, página 1137–1145. Citado na pág. 12, 23, 28

Leite e Macedo(2017) C. Leite e M. Macedo. Corrupcao politica: a colonizacao do brasil. *Periódico Científico Outras Palavras*, 13:109–115. Citado na pág. 1

Lundberg(2018) S. Lundberg. SHAP. https://shap.readthedocs.io/, 2018. [Online; accessed 12-September-2019]. Citado na pág. 15

Lundberg e Lee(2017) S. Lundberg e S. Lee. A unified approach to interpreting model predictions. páginas 4765–4774. URL http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf. Citado na pág. 15

Pérez(2015) F. Pérez. Project Jupyter. https://jupyter.org/, 2015. [Online; accessed 27-June-2019]. Citado na pág. 3

Ransom(2013) J. Ransom. Replicating data mining techniques for development: a case study of corruption. Dissertação de Mestrado, Master of Science, International Development and Management, Lund University, Sweden. Citado na pág. 1

Resende e Andrade(2011) J. Resende e M. Andrade. Crime social, castigo social: desigualdade de renda e taxas de criminalidade nos grandes municípios brasileiros. *Estudos Econômicos, São Paulo - SP*, 41:174–195. Citado na pág. 1

Rossum(2001) G. Rossum. Python Enhancement Proposal 8. https://www.python.org/dev/peps/pep-0008/, 2001. [Online; accessed 27-June-2019]. Citado na pág. 3

Rossum(1990) G. Rossum. Python programming language. https://www.python.org/, 1990. [Online; accessed 27-June-2019]. Citado na pág. 3

Santosa e Symes(1986) F. Santosa e W. Symes. Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing*, páginas 1307–1330. Citado na pág. 17

Shapley e Tucker(1953) Kuhn H. Shapley, L. e A. Tucker. *Contributions to the Theory of Games*. Citado na pág. 14

Shinyama(2004) Y. Shinyama. PDFMiner. https://euske.github.io/pdfminer/index.html, 2004. [Online; accessed 27-June-2019]. Citado na pág. 5

Silva e Carvalho(2012) M. Silva e P. Carvalho. Building a Sentiment Lexicon for Social Judgement Mining. *Proceedings of the 10th international conference on Computational Processing of the Portuguese Language*, páginas 218–228. Citado na pág. 5, 31

Spärck Jones(1972) K. Spärck Jones. A Statistical Interpretation of Term Specificity and Its Application in Retrieval. *Journal of Documentation*, páginas 11–21. Citado na pág. 6

Tibshirani(1996) R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society*. Citado na pág. 17

Wesllen(2013) D. Wesllen. Diagnostico de influencia bayesiano em modelos de regressao da familia t-assimetrica. Dissertação de Mestrado, Mestrado em Estatística, Instituto de Matematica e Estatistica, Universidade de Sao Paulo, Brasil. Citado na pág. 12