Análise de Redes Sociais com Uso de Aprendizado de Máquina para Prever o Tráfego de Veículos em Zonas Urbanas

Trabalho de Conclusão de Curso apresentado ao
Departamento de Ciência da Computação do
Instituto de Matemática e Estatística da
Universidade de São Paulo

Lucas de Carvalho Dias
Orientador: Roberto M. Cesar Jr

São Paulo, 2017

Este trabalho contou com apoio financeiro e institucional de Iniciação Científica da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP). Processo: 2017/00406-8. Agradecemos também ao projeto Temático FAPESP 2015/22308-2.

Resumo

Em 2015, eram publicados cerca de 350 mil *tweets* por minuto. Estes continham informações sobre eventos e fatos que ocorreram no cotidiano dos usuários. Por incluírem parâmetros geográficos e temporais, essas publicações podem ser relacionadas com dados de trânsito, de modo a extraírem padrões que relacionem os dois tipos de parâmetros indicados. O objetivo deste projeto é desenvolver e implementar um método que usa a fusão desses dados em algoritmos de aprendizado de máquina, para, dessa forma, identificar automaticamente se um *tweet* está informando sobre o trânsito e, caso o esteja, a que tipo de evento o texto da publicação diz respeito. Em um período de duas semanas, foram coletados cerca de 300 Gigabytes de eventos de tráfego de veículos na cidade de São Paulo, e 10 Gigabytes de publicações do Twitter geo-localizadas na mesma cidade. Os dois tipos de dados foram filtrados e formatados para poderem ser fundidos. No resultado da fusão, foram empregadas técnicas de processamento de linguagem natural, com objetivo de prepará-lo para alimentar as rotinas de aprendizado de máquina implementadas.

Palavras-chave: Informática Urbana, Fusão de Dados, Classificação de Texto, Redes Sociais, Aprendizado de Máquina.

Abstract

In 2015, 350 thousand tweets were published every minute. These contained information about events and facts that occured in the users daily life. By having geografical and temporal parameters, these publications can be connected to vehicle traffic data, so that patterns of the relation between the two kinds of parameters are extracted. The goal of this project is to develop and implement a method that uses the data fusion of such data in machine learning algorithms in order to automatically identify if a tweet is informative about vehicle traffic and, if it is, to which kind of event the posted text is reffering to. In a period of two weeks, an amount of 300 Gigabytes of vehicle traffic event data in the city of Sao Paulo were collected, and 10 Gigabyte of Twitter posts geografficaly localized in the same city. Both datasets were subjected to filtering and formmating to be able to be fused. With the result of the fusion, natural language processing techiniques were applyied with the objective of preparing it to feed the machine learning scripts implemented.

Keywords: Urban Informatics, Data Fusion, Text Classification, Social Networks, Machine Learning.

Sumário

Lista de Figuras

Parte I

Parte Objetiva

Capítulo 1

Introdução

1.1 Motivação

Nas grandes cidades, localizam-se milhões de elementos que se movimentam e interagem de forma dinâmica. Tal conjunto gera uma gigantesca máquina complexa, caótica e heterogênea, assim como se verifica com os dados que o representam. Estes, que incluem meta-dados, carregam informações sobre o que ocorre a todo momento pela cidade. Devido à natureza complexa desses dados, torna-se difícil a análise para a busca de padrões que nos forneçam alguma previsibilidade sobre o que está por vir.

Felizmente, com o nível de tecnologia atual, já é possível lidar com esse volume de dados empregando técnicas computacionais. Esta abordagem é conhecida como informática urbana (*urban informatics*), usada para extrair, a partir desta grande quantidade de informações, compreensão sobre aspectos inerentes ao funcionamento das cidades, tais como mobilidade urbana, violência, economia, etc. De fato, esta já é uma estratégia de interesse de muitas prefeituras. Kitchin [20], em um artigo sobre cidades inteligentes (*Smart Cities*), cita o exemplo do Centro de Operações da prefeitura do Rio de Janeiro, uma parceria do governo municipal e da IBM que tem como tarefa investigar dados obtidos de trinta agências (dados de trânsito, serviços de emergência, previsão do tempo, etc.). Estes são analisados, com uso de algoritmos, em busca da compreensão de aspectos da vida na cidade e na construção de modelos de previsão.

A área de informática urbana tem se desenvolvido intensamente nos últimos anos. Técnicas de ciência de dados estão sendo criadas para analisar problemas como violência urbana [13], trânsito intenso [30], acidentes de trânsito [14] e gasto energético [27].

Nesse sentido, atualmente, um dos principais problemas que assola a vida dos moradores das grandes cidades é o congestionamento nas vias principais. A análise de dados de redes sociais se tornou um tópico de pesquisa bastante rico e complexo que pode ser usada para extrair conhecimento sobre diversos comportamentos sociais [12]. Chong *et al.* [14] usou um conjunto de dados gerado a partir de amostras coletadas de acidentes de trânsito nos EUA para classificar a severidade dos acidentes aplicando redes neurais, árvores de decisão, SVM (Support Vector Machine) e uma combinação

híbrida de redes neurais e árvores de decisão. Wang *et al.* [31] propõe um sistema de alerta de tráfego que usa LDA(*Latent Dirichlet Allocation*) para classificação de tópicos dos textos de *tweets* relacionados a trânsito. Ferrari *et al.* [16] usa parâmetros geo-temporais de *tweets* da cidade de Nova Iorque para detectar multidões e aplica *topic mining* probabilístico no texto desses *tweets* para detectar eventos que estão relacionados ao cenário urbano. Cranshaw *et al.* [15] particiona (*cluster*) *check-ins* na rede social *Foursquare* com métricas baseadas em coordenadas geográficas e na similaridade do conjunto de usuários que visitam cada lugar. Dessa forma, seu método leva em conta tanto a proximidade espacial das *venues* (estabelecimentos) quanto a proximidade social, derivada da distribuição de pessoas que fazem *check-in* nelas. Pan *et al.* [25] aborda o problema de identificar anomalias de tráfego usando *crowd sensing* com dados de mobilidade e redes sociais. Uma anomalia detectada é representada como um sub-grafo de uma rede de avenidas em que a rota de um motorista diverge consideravelmente de seu padrão usual. Para descrevê-la, são minerados da rede social termos que foram postados quando esta ocorreu.

Como alguns dados de redes sociais possuem a informação do local de onde foram publicados, eles podem ser empregados, a fim de se obter informações sobre a mobilidade na cidade. Também é sabido que chuvas influenciam bastante a qualidade do trânsito, de forma que dados meteorológicos podem fornecer informações úteis sobre a mobilidade.

Motivado por esses fatos, este trabalho objetiva estudar de que modo os dados de redes sociais e meteorológicos expressam informações sobre o estado do trânsito. Para tal, serão coletados dados a partir de redes sociais, aplicações de trânsito e fontes de notícias virtuais. Após serem formatadas e filtradas, estas informações servirão como conjunto de dados para algoritmos de *data mining* e aprendizado de máquina (*machine learning*). Quando estas técnicas forem aplicadas, serão obtidas conclusões sobre como o trânsito é afetado por diversos eventos que ocorrem na cidade (sejam eles de origem humana ou natural). Estas nos fornecerão também padrões que podem ser empregados em conjunto a técnicas de aprendizado de máquina com o intuito de fazer previsões sobre como eventos futuros interferirão no fluxo de trânsito.

1.2 Objetivo

O objetivo deste projeto é estudar e desenvolver técnicas de mineração de textos (*text mining*) e aprendizado de máquina capazes de relacionar dados urbanos de fontes

heterogêneas, bem como obter tal conjunto de dados.

1.3 Organização do Trabalho

No capítulo que segue (Capitulo 2 - Fundamentos Teóricos) é feita uma breve explicação de conceitos e técnicas utilizados no trabalho, como Classificação de Texto na seção 2.1 e métodos para aplicá-la (Máquinas de Vetor de Suporte 2.4.3, Redes Neurais Convolucionais 2.4.2), Redução de Dimensionalidade e Visualização de Características 2.3, e outros. O desenvolvimento do projeto é apresentado no capítulo 3 que inicia explicando o procedimento de coleta dos dados de diferentes fontes 3.1, seguido pelo processo de tratamento dos mesmos 3.2 e a fusão de todos os conjuntos em um 3.3. Continua-se o capítulo apresentando a decodificação dos dados textuais 3.4 para então explicar como serão inseridos em um processo de aprendizado de máquina 3.5, e os resultados obtidos ao aplicar a metodologia exposta finaliza o capítulo. No capítulo 4 são descritas as conclusões obtidas acerca do trabalho realizado.

Capítulo 2

Fundamentos Teóricos

2.1 Classificação de Texto

Classificação de texto corresponde ao problema de atribuir classes ou categorias predeterminadas para textos. Abordar tal problema computacionalmente é tratá-lo como uma questão de Processamento de Linguagem Natural visto que para suceder na tarefa é necessário o reconhecimento de padrões no texto. Uma das formas para abordar esse problema é o uso de Aprendizado de Máquina de forma que é aplicado um algoritmo que aprende padrões analisando uma grande quantidade de textos pertencentes a um conjunto de dados predefinido.

Sebastianie [28] define o problema matematicamente da seguinte forma:

Definição 1: Classificação de texto é a tarefa de atribuir um valor booleano para cada par $< t_i, c_j > \in TxC$, sendo T um domínio de textos e $C = [c_1, ..., c_{|C|}]$ é um conjunto de categorias pré-definidas. Um valor V atribuído a $< t_i, c_j >$ indica uma decisão para o texto t_i sob c_j, enquanto um valor F indica uma não decisão para t_i sob c_j. Então o problema se resume a aproximar uma função $\Phi' : DxC \rightarrow V, F$ a partir de uma função $\Phi : DxC \rightarrow V, F$ chamada classificador (modelo).

2.2 Codificação de Dados

Os algoritmos de aprendizado de máquina são ferramentas matemáticas e, por isso, dados que não se encontram naturalmente em forma numérica precisam ser codificados como tal. Em função disso, para alimentar tais algoritmos com textos faz-se necessário o uso de técnicas de codificação neste conjunto. Algumas destas técnicas são descritas a seguir.

2.2.1 N-Gramas

Em algumas aplicações de classificação de texto a melhor forma de abordar o problema não é classificar o texto inteiro, e sim dividir ele em pequenas partes de n termos para classificar cada uma delas. Essa abordagem é útil quando se quer identificar a presença ou ausência de elementos subjetivos (como um endereço ou emoção) ao longo do texto ou prever o próximo termo dado os termos anteriores.

2.2.2 *word2vec*

Introduzida por Mikolov et al. em [24], é uma técnica que consiste em treinar um modelo de rede neural denominado *Skip-Gram* que calcula a probabilidade de cada palavra aparecer ao redor de uma dada palavra. A camada escondida da rede neural é extraída e usada como codificador (mapeamento de palavra para um vetor).

2.3 Redução de Dimensionalidade e Visualização das Características

2.3.1 Análise de Componentes Principais (Principal Component Analysis - PCA)

PCA, desenvolvido por [18], trata-se de uma técnica que faz uma transformação ortogonal na matriz de dados de forma que as coordenadas estejam ordenadas crescentemente da que contém a maior variância quando projetados os pontos da matriz nela até a que contém menor variância.

Usando essa técnica nos vetores de características tem-se projeções dos dados que estão organizadas da mais relevante para a menos relevante. Com isso é possível escolher apenas algumas das projeções mais relevantes para-se usar no futuro, fazendo, assim, uma diminuição da dimensionalidade do vetor de características.

É possível usar Decomposição de Valor Singular (*Singular Value Decomposition* - SVD, mostrado como calcular por Golub e Reinsch em [17]) para executar a análise de componentes principais. O processo é explicado por Shlens [29] do seguinte modo:

Tendo uma matriz X $m \times n$ sendo o conjunto de dados. Seja $Y \equiv \frac{1}{sqrtn} X^T$. Temos que $Y^T Y = (\frac{1}{sqrtn} X^T)^T (\frac{1}{sqrtn} X^T) = \frac{1}{n} X X^T = C_X$, ou seja $Y^T Y$ é igual a matriz de covariância de X. Então os componentes principais de X são os auto-vetores de C_X. Calculando o SVD de Y (obtendo $Y = U \Sigma V^T$), as colunas da matriz V conterão os auto-vetores de C_X. Dessa forma, as colunas de V são os componentes principais de X.

2.3.2 t-SNE

t-Distributed Stochastic Neighboor Embbeding, desenvolvido por Mateen e Hinton em [23], tem o mesmo objetivo do *PCA*: diminuir a dimensão dos dados de forma que se perca o mínimo possível da expressividade de suas características. Consiste em calcular, para cada ponto (amostra do conjunto de dados), a distribuição da posição dos pontos a sua volta (seus vizinhos) no espaço de alta dimensionalidade. E tenta, em um espaço de baixa dimensionalidade, dispor os pontos de forma que todas as distribuições continuem o mais próximo possível das originais (no espaço de alta dimensionalidade).

2.4 Classificadores

2.4.1 Redes Neurais Artificiais(*Artificial Neural Networks* - ANN)

Uma rede neural artificial é uma estrutura baseada na biologia, é composta por unidades denominadas neurônios que recebem um vetor de características como entrada, multiplicam-nos com um vetor de pesos e somam com um número de viés, como é ilustrado na Figura 2.1. O resultado é então submetido a uma função não linear e devolvido como saída da unidade. Ou seja, cada unidade faz a operação $\sigma(Wx + b)$.

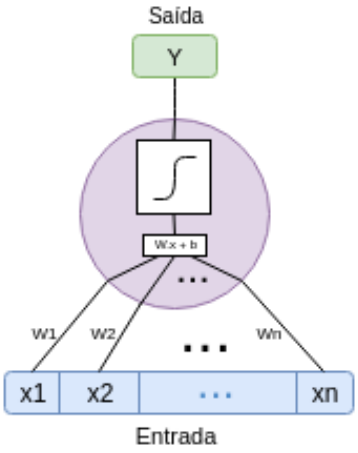

Figura 2.1: Unidade de uma Rede Neural Artificial.

Essas unidades são organizadas em camadas, e a saída dos neurônios da camada anterior são dadas como entrada para cada unidade da próxima camada. Para treinar os pesos (W) e os viéses (b) da rede são usados algoritmos que comparam as saídas da rede com as saídas de um conjunto rotulado através de uma função de erro (*loss*

function), e achando a direção para levar os pesos que acarreta em um mínimo nesta função. O algoritmo mais usado é a propagação reversa (*backpropagation*) desenvolvido por Werbos em [32].

2.4.2 Redes Neurais Convolucionais(*Convolutional Neural Networks* - CNN)

Redes Neurais Convolucionais, como definido por LeCun *et al.* em [21], são redes neurais com arquitetura feita da combinação de:

- Campos receptivos locais. Ou seja, ao contrário de uma rede neural clássica, unidades com essa característica não enxergariam toda a entrada, mas sim um pedaço local desta.;

- Pesos compartilhados (ou replicação de pesos);

- Sub-amostragem.

Dessa forma, as CNN's geralmente são compostas dos seguintes tipos de camadas:

- Convolucional: Cada unidade, denominada filtro, recebe valores de um pedaço da entrada correspondente a uma janela (seu campo receptivo).

- Sub-amostragem (*Pooling*): Nesta camada, a função da unidade é selecionar a melhor saída dentre saídas das unidades da camada anterior que estão dentro do campo receptivo da unidade atual;

- Totalmente Conectada (*Fully-Connected*): Essa camada segue a arquitetura das redes neurais clássicas.

2.4.3 Máquina de Vetor de Suporte(*Support Vector Machine* - SVM)

Enxergando as amostras de um conjunto de dados como pontos em um espaço, a Máquina de Vetor de Suporte (Ilustrada na Figura 2.2) usa programação quadrática para encontrar uma fronteira que divide o espaço em duas partes, de forma que a soma das distâncias entre os pontos e a fronteira seja a maior possível.

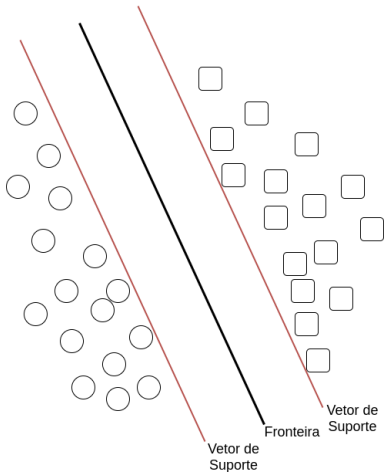

Figura 2.2: Exemplo de SVM.

2.5 Expressões Regulares

Expressões regulares são um código que permite representar padrões de sequências de caracteres. Com esse código é possível executar buscas em textos e obter todas as sequências que satisfazem o padrão escolhido.

Segundo Alfred e Ullman [11], Expressões Regulares podem ser definidas em termos de autômatos finitos como:

Definição 2: Seja um alfabeto finito σ, $\emptyset$ é uma expressão regular vazia (não é pareada com nenhum texto) e e é uma expressão regular que denota o conjunto contendo o caractere nula como único elemento. Então temos o seguinte:

- $a \in \sigma$ é uma expressão regular denotando o conjunto a;
- Se p e q são expressões regulares denotando os conjuntos P e Q, respectivamente, então:
  - $(p + q)$ é uma expressão regular denotando o conjunto $P \cup Q$;
  - (pq) é uma expressão regular que denota o conjunto composto por todas as concatenações dos elementos de P com os elementos de Q;
  - $p*$ é uma expressão regular que representa todas as concatenações de zero ou mais elementos de P.

Capítulo 3

Experimento

O processo proposto neste trabalho é dividido em duas partes: Coleta de Dados e Análise dos Dados coletados. Para realizar a primeira etapa do trabalho foram implementados, para cada fonte de dados, rotinas que ficariam em execução durante o período de coleta enviando requisições (usando suas respectivas *API's*) em intervalos de tempo para estas fontes e armazenando as respostas.

O conjunto gerado foi então submetido a um processo de tratamento que envolve filtragem e formatação. Como as requisições retornavam arquivos JSON (*JavaScript Object Notation*) [4], o *dataset* primeiro foi convertido para um formato de tabela (*Comma Separeted Value* - CSV) [1]. Após a conversão as entradas duplicadas foram eliminadas, e os meta-dados que não seriam usados foram descartados.

Para completar o conjunto de dados dos *tweets* com parâmetros de localização precisos, foram empregados dois métodos para obter os endereços dos textos dos *tweets* e os converte para coordenadas geográficas. O método que apresentou melhor resultado foi empregado para completar o *dataset*.

O papel dos dados de trânsito é servir de rótulo para os *tweets*. Para isso é feita uma fusão dos *datasets* de *tweets* e incidentes. Para cada *tweet*, este é emparelhado com o incidente ao qual corresponde melhor.

Com o *dataset* completo, ainda é preciso transformar dados textuais em numéricos para que possam ser alimentados ao processo de aprendizado de maquina. Então é feito um processo de codificação e extração de características. Enfim, o *dataset* finalizado seria usado para treinar um classificador de *tweets* de forma supervisionada.

3.1 Coleta dos Dados

Houve dois períodos de coleta de dados. No primeiro, foram coletados dados (trânsito, postagens e meteorologia) da região que corresponde ao estado de São Paulo. No segundo período, trata-se da região que corresponde à região metropolitana da cidade de São Paulo.

Foram executadas duas coletas, pois a primeira gerou uma quantidade de dados que tornaria difícil até manipulações básicas (como mover o arquivo no computador). Por isso, foi também efetuada uma coleta, desta vez com maior restrição do parâmetro espacial para englobar somente a área da região metropolitana da cidade de São Paulo em uma segunda coleta. Portanto, para as etapas do seguintes trabalho foram selecionados para uso apenas os dados provenientes da segunda coleta, conforme é especificado a seguir.

3.1.1 Fontes de Dados

– twitter.com [10]:
Utilizando a streaming API - API disponibilizada pelo twitter para aplicações que mineram grandes quantidades de dados continuamente. Foi usada para obter os textos que foram twittados na região do estado de São Paulo. Limite: Não se aplica.
– here.com [3]:
Usando a Traffic Flow API - Foram obtidos dados da fluência do trânsito no estado de São Paulo. Usando a Traffic Incidents API - A partir dela, foram obtidos dados dos incidentes de trânsito no estado de São Paulo. Limite: 100.000 transações por mês.
– openweathermap.org [5]:
Usada a Current Weather Data API - Foram obtidos dados meteorológicos (temperatura, pressão, velocidade do vento, etc) das cidades da região metropolitana da cidade de São Paulo. Limite: 60 requisições por minuto.

3.1.2 Períodos de Coleta

Os períodos das coletas foram:
– Primeira:
  • Início: 20:11 Sexta-Feira 28/04/2017 Horário de Brasília.
  • Fim: 1:59 Terça-Feira 09/05/2017 Horário de Brasília.
– Segunda:
  • Início: 11:01 Quarta-Feira 7/06/2017 Horário de Brasília.
  • Fim: 2:02 Quarta-Feira 21/06/2017 Horário de Brasília.

3.1.3 Especificações

– twitter.com:

- *Tweets* (textos com 143 caracteres) postados nos últimos instantes para um dado instante, cujos usuários estavam dentro da *bounding box* do estado de SP quando postaram.
- Bounding Box:
  - Primeira: -53.1096, -25.2505, -44.1606, -19.7793.
  - Segunda: -46.8254, -24.0084, -46.3648, -23.3576.
- Requisição: As requisições são mediadas pela *Streaming API* do *Twitter*. Assim que é estabelecida a conexão e enquanto esta se mantiver ativa, são mandados pelo twitter *tweets*, um por vez. Cada *tweet* é dado como entrada para uma função definida no *script* para tratar a resposta. No caso, a função adicionava a nova resposta a um arquivo JSON.
- Resposta: Cada resposta continha os dados de um *tweet* em JSON. Exemplo:

– here.com:

- *Flow* : Lista de Fatores de Congestionamento das vias dentro do *bounding box* em dado instante. Incidents: Lista de Incidentes que ocorreram nos últimos minutos para dado instante. Contém a gravidade do incidente, uma pequena descrição em texto e a localização. As vezes contém informação da faixa em que ocorreu.
- Bounding Box:
  - Primeira: -53.1096, -25.2505, -44.1606, -19.7793.
  - Segunda: -46.8254, -24.0084, -46.3648, -23.3576.
- Requisição:
  - -Flow: De 10 em 10 minutos foram feitas requisições que retornassem as todas as reportagens de fluidez na região metropolitana de São Paulo, no horário da requisição, de uma vez.
  - -Incidents: Para obter incidentes de toda a região metropolitana de São Paulo, no máximo de 10 em 10 minutos para não ultrapassar o limite de requisições, usou-se uma requisição que retornasse todos os incidentes ativos da região toda de uma vez.
- Resposta:
  - -Flow: Todas as reportagens do estado da fluência do trânsito, no horário da requisição, na região dada (*bounding box* da região metropolitana de São Paulo).
  - -Incidents: Todos os incidentes ativos, no horário da requisição, na região

dada (*bounding box* da região metropolitana de São Paulo).

– openweathermap.org: Para cada cidade na região metropolitana, tem-se a temperatura, umidade, pressão, velocidade do vento, etc. medidos nos últimos 10 - 30 minutos.

- Requisição: Mandava-se requisições que retornassem a medição de todas as cidades da região metropolitana de São Paulo, no dado momento.
- Resposta: A medição de temperatura, umidade, velocidade do vento, direção do vento e estado das nuvens de todas as cidades da região metropolitana de São Paulo, no dado momento.

3.1.4 Implementação

Foi usado um *script* para a coleta de cada fonte de dados, todos construídos em cima de uma classe geral de mineração que também foi implementada. O funcionamento deles se baseia em ficar em espera por 10 minutos, acordar, fazer uma requisição, armazenar a resposta e voltar a ficar em espera.

Como a rotina em Python da mineração deve permanecer em execução durante todo o período de coleta, é necessário que o programa seja implementado de forma a ser à prova de erros, pois não há possibilidade de monitorar o funcionamento do programa, e qualquer erro que acontece o tempo que se leva para perceber que o programa parou pode ser grande. Isso é perigoso, já que pode gerar buracos no *dataset* que podem comprometer sua completude, ainda mais quando se trata de programas que dependem de requisições HTTP. Para evitar problemas, além de implementar o programa de forma que conseguisse lidar com diversos erros de conexão, também foi implementado um sistema para avisar por *email* qualquer evento de importância que ocorria com o programa.

3.2 Tratamento dos dados

Para construir um *dataset* a partir da integração de dados de diferentes fontes é necessário que eles estejam em um mesmo formato. O processo de padronização envolve executar buscas e filtragens nas tabelas para alinhar as informações com base em seus parâmetros espaciais e temporais, e para fazer isso de forma efetiva, é necessário que toda a tabela possa ser aberta por um programa em execução, o que não é possível para arquivos de dezenas de *gigabytes*(caso do arquivo que contém os dados de incidentes e

fluxo).

A etapa inicial da da padronização consiste em fragmentar todo o conjunto de dados em vários (ou diversos) arquivos, cada um contendo entradas de um mesmo dia do período de coleta. A rotina de conversão é executada em cada um dos fragmentos e isso permite aproveitar o uso de vários núcleos do processador para executar todos os fragmentos paralelamente. Depois de processar os fragmentos são mesclados, e uma nova filtragem de duplicados é feita, caso dois fragmentos contenham a mesma entrada.

Então, executa-se a padronização dos conjuntos para a formação de um *dataset* que pode ser inserido facilmente em algoritmos de Aprendizado de Máquina e Mineração de Dados. O processo de padronização consiste em uma etapa de extração de informação dos dados textuais para construir uma tabela; depois, uma etapa de filtragem, quando são removidas entradas duplicadas e são descartadas informações irrelevantes. Todas as operações nas tabelas são feitas com funções da biblioteca Pandas [6], que implementa funções de busca e filtragem eficientes.

3.3 Fusão de Dados

Para rotular os *tweets* coletados, é feita uma fusão destes com os dados de incidentes de trânsito. Para cada amostra do conjunto de dados da rede social, é preciso encontrar nos dados de incidentes qual melhor corresponde a ele. Os principais parâmetros que devem ser alinhados são o de tempo e espaço. O *tweet* precisa estar se referindo ao mesmo local em que ocorreu o incidente, e precisa ter sido postado próximo do período em que o incidente ocorreu.

No entanto, observando os meta-dados dos *tweets* coletados, é possível notar que, em sua maioria, o parâmetro da localização é impreciso, geralmente apenas indicando a cidade de onde foi postado. O local, com mais exatidão, é escrito por extenso no texto. E também há o problema de quando alguém posta sobre algo após o evento, não se encontrando mais no local do ocorrido.

Como para realizar a fusão dos *datasets* do Twitter e dos Incidentes de trânsito um dos parâmetros mais importantes é a geo-localização, faz-se necessário deduzir o local a que o texto se refere a partir do próprio texto, e não usando a informação indicada nos meta-dados. Dessa forma, foram empregados dois métodos que abordam esse problema de formas diferentes, e o método que apresentou melhor resultado foi utilizado para realizar a tarefa.

O primeiro método consiste em abordar como um problema de classificação de texto.

O segundo método faz uma medição de similaridade entre o texto do *tweet* e a descrição de cada amostra de incidente de trânsito, sendo, assim, paralelo à etapa da fusão dos dados.

3.3.1 Primeiro Método - Classificação

Para encaixar(ou adaptar) o problema de extrair um endereço como um problema de aprendizagem de máquina, pode-se modelá-lo da seguinte forma: para cada pedaço de texto com n tokens (ou seja, para cada n-grama), o classificador deve ser capaz de identificar se este contém um (ou parte de um) endereço. É possível também melhorar a exatidão, além de identificar se tem ou não, pode também identificar se é o começo, meio ou fim de um endereço. Observando amostras de *tweets*, como na Figura 3.1, é notável que muitas vezes o endereço está fragmentado ao longo do texto. Isto ocorre mais frequentemente quando o endereço é em uma rodovia, e contém o número do quilometro.

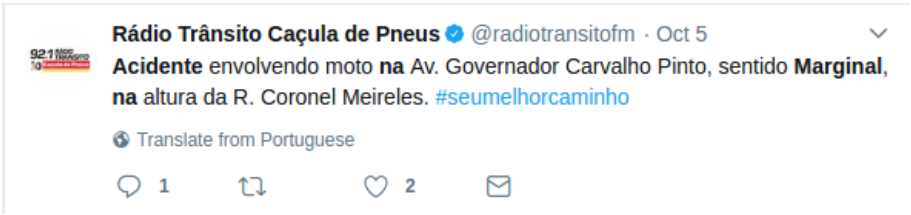

Figura 3.1: Exemplo de *tweet*, com um endereço descrito textualmente mas sem parâmetros de localização geográfica.

3.3.1.1 Seleção para o *dataset* de Endereços

Para compor o conjunto de dados foram selecionadas as seguintes contas do twitter:

- radiotransitofm;
- ZeroTransitoSP;
- TaxiAlphaville.

Foram extraídos 1000 postagens de cada, totalizando 3000 postagens. Em cada postagem, foi feita a separação em 5-gramas, resultando em 38007 5-gramas. Esses dados foram organizados em formato de tabela, contendo as seguintes colunas:

- 5-grama;
- Texto original do *tweet*;
- índices correspondentes ao 5-grama no texto;
- Rótulo.

Foi escolhido o valor dos n-gramas como $n = 5$ para que na maioria dos casos coubesse um endereço inteiro em um n-grama e ao mesmo tempo não ficasse um pedaço muito grande que tornasse imprecisa a extração da parte que corresponde ao endereço de fato.

3.3.1.2 Rotulação

Em razão de que rotular manualmente um conjunto de dados com dezenas de milhares de entradas pode demorar um tempo inábil, foi feita uma pré-rotulação usando uma Expressão Regular:

```
(\p{Lu}\p{L}+(?:\s\p{Lu}\p{L}+|\s\p{Ll}{1,2}(?=\s\p{Lu}))+
(?: comma\s\d+)?)
```

A expressão funciona da seguinte maneira:

- `\p{Lu}`emparelha com letra maiuscula;
- `\p{Ll}`emparelha com letra minúscula;
- `\p{L}`emparelha com qualquer letra;
- `\p{Lu}\p{L}+`emparelha com letra maiuscula e depois uma ou mais letras quaisquer;
- `(?:\s\p{Lu}\p{L}+|\s\p{Ll}{1,2}(?=\s\p{Lu}))+`emparelha se houver um ou mais espaço seguido de Letra Maiúscula seguida de letra qualquer ou espaço seguido de uma ou duas letras minúsculas seguidas de espaço e uma letra maiúscula;
- `(?:comma\s\d+)?` Opcionalmente, emparelha se houver a palavra *comma* (que representa uma vírgula no texto original) seguida de espaço e um número.

Após executada a extração com a regex, os 5-gramas relacionados a cada endereço eram processados para verificar se continham o começo, meio, fim ou um túnel (se é

o fim parcial de um endereço e ele começa de novo em outro 5-grama, ou se está começando de novo após um fim parcial e texto que não pertence ao endereço) do endereço e, então, o respectivo (pré-)rótulo era dado. Ao extrair 5-gramas de textos as pontuações são removidas, mas como a vírgula pode ser importante em endereços, elas foram substituídas pela palavra *comma*. Rótulos dos 5-grams:

- NOT = 0 - não contém um endereço;
- BEGIN = 1 - contém o começo de um endereço (no fim de seu texto);
- MIDDLE = 2 - contém o meio de um endereço (todo seu texto é parte do endereço);
- END = 3 - contém o fim de um endereço (no começo de seu texto);

Feita a pré-rotulação, exemplificada na Tabela 3.1, a rotulação de fato foi apenas uma questão de passar pela tabela, corrigindo os eventuais erros.

Rótulo	5-grama
1	ATUALIZAÇÃO SP148 Rod Cam do
2	SP148 Rod Cam do Mar
3	Rod Cam do Mar ainda
3	Cam do Mar ainda engarrafada
3	do Mar ainda engarrafada Somará
3	Mar ainda engarrafada Somará 42m
0	ainda engarrafada Somará 42m ao
0	engarrafada Somará 42m ao seu
0	Somará 42m ao seu tempo
0	42m ao seu tempo de
0	ao seu tempo de deslocamento
0	seu tempo de deslocamento seumelhorcaminho
0	tempo de deslocamento seumelhorcaminho transito

Tabela 3.1: Exemplo de rotulação para extração de endereço do texto.

3.3.1.3 Validação e Geocoding

Mesmo após extrair o endereço do texto, ele se encontra em formato de texto, o que impossibilita comparações diretas entre endereços para tirar conclusões acerca de proximidade. Para transformar um endereço em formato de texto em coordenadas geográficas usou-se a biblioteca do Python Geocoder [7], chamando a função que faz

requisições para o Google Maps que, ao fazer (*geocoding*), retornava as coordenadas do endereço e a *bounding box* (caso o endereço não fosse muito especifico e.g: uma rua sem especificar a numeração).

Ao fazer *Geocoding* de algumas entradas, a ferramenta não era capaz de encontrar o endereço ou encontrava o endereço errado. Para filtrar esses casos, só eram adicionados no novo *dataset* entradas que continham coordenadas que estavam dentro da *bounding box* escolhida na etapa de coleta de dados.

3.3.1.4 Classificação

Para a classificação usou-se exatamente o mesmo processo descrito na seção de Classificação de Incidentes. As únicas diferenças são a entrada, que são 5-gramas ao invés de um texto completo, e os valores dos rótulos.

3.3.1.5 Fusão

Tendo feito o treinamento do classificador, este é usado para prever o rótulo de cada 5-grama de todo o conjunto de dados de *tweets*. Com os 5-gramas rotulados, é possível localizar, no texto, onde foi previsto a presença de um endereço e, assim, completar o *dataset* com parâmetros geográficos mais precisos.

3.3.2 Segundo Método - Similaridade de Palavras

No conjunto de dados de incidentes de trânsito, um dos meta-dados é uma descrição textual que explica o que aconteceu e especifica mais onde aconteceu. Nos *tweets* que relatam um incidente, o local está geralmente contido no texto também. Dessa forma, é possível identificar se um *tweet* diz respeito a uma entrada presente no conjunto de dados de incidentes através da similaridade entre seu texto e o do endereço concatenado com a descrição detalhada do incidente. Para calcular a similaridade, foi usada a distância Levenshtein (definida por Levenshtein e traduzida para o inglês por Novikov em [22]), conhecida como distância de edição. Seu calculo é baseado no número de caracteres que precisam ser alterados para uma palavra se tornar equivalente a uma outra.

A distância Levenshtein entre dois textos a, b é dada por $lev_{a,b}(|a|, |b|)$, onde:

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j), & \text{if } \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{, c.c} \end{cases}$$

É calculada a distância de Levenshtein entre cada palavra do texto do *tweet* e cada palavra do resultado da concatenação entre a descrição e o endereço do incidente. Se a soma das N palavras com menor distância for menor que D e o momento em que o *tweet* foi postado não difere do período do incidente mais do que M minutos, o incidente em questão é considerado candidato para ser emparelhado com o *tweet*. Os valores escolhidos experimentalmente são mostrados na tabela 3.2. Em seguida, a lista de candidatos é ordenada de acordo com o tanto que diferem no tempo e na similaridade do texto, e o que apresentar menor pontuação é selecionado.

Parâmetro	Valor
M	30
N	10
D	13

Tabela 3.2: Parâmetros da Fusão escolhidos experimentalmente

3.4 Codificação

Algoritmos de Aprendizado de máquina são ferramentas matemáticas e, por isso, é necessário que a entrada(*input*) do processo esteja em formato numérico. É feito um processamento do texto para produzir um vocabulário com todas as palavras presentes em todos os textos do conjunto de dados. Com esse vocabulário, faz-se um mapeamento de cada palavra para um número. Com este, transforma-se os textos para uma lista de números.

Quando o conjunto de dados a ser usado para treinamento não se encontra naturalmente em formato numérico, é necessário fazer uma conversão. Muitas vezes é uma tarefa simples. Se as características a serem usadas são conhecidas e estão explicitas nos dados, elas podem receber arbitrariamente um numero de identificação.

O problema com essa codificação é que ela não leva em consideração a similaridade entre palavras. Por isso uma palavra pode acabar próxima vetorialmente de outra completamente diferente, e distante de um sinônimo. Para obter uma representação que considera a similaridade entre as palavras foi usada a técnica *word2vec*. Para treinar tal modelo foi usada a biblioteca *gensim* [26] do *Python*. A corpora usada foram os própios textos dos *tweets coletados*.

3.5 Classificação de Tweets

A classificação dos incidentes relatados nos *tweets* é feita com dois métodos para comparar duas técnicas comumente usadas em problemas de classificação.

A primeira técnica utiliza uma rede neural com uma camada de convolução, seguida de uma de *max-pooling* e uma totalmente conectada (*fully-connected*). O resultado da predição é obtido aplicando a função exponencial normalizada(*softmax*) no resultado da ultima camada.

Softmax:

$$\sigma(z)_j = \frac{e^{x^{z_j}}}{\sum_{k=1}^{K} e^{x^{z_k}}}$$

A função de custo utilizada para avaliar a atualização dos pesos da rede é *cross-entropy*, definida como:

$$H_{y'}(y) = -\sum_i y'_i \log y_i$$

Onde y é a distribuição observada e y' é a distribuição real segundo a rotulação. E o algoritmo usado para atualização dos pesos é o Adam [19].

Já a segunda técnica é uma Máquina de Vetor de Suporte (SVM - *Support Vector Machine*). É feita uma busca em grade (*GridSearch*), com seu processo explicado no diagrama da Figura 3.2 para encontrar os melhores valores para γ entre $[1, 5, 10, 15]$ e C, com os mesmos valores testados. Todos os valores foram testados com o *kernel* linear e o RBF(*Radial Basis Function*).

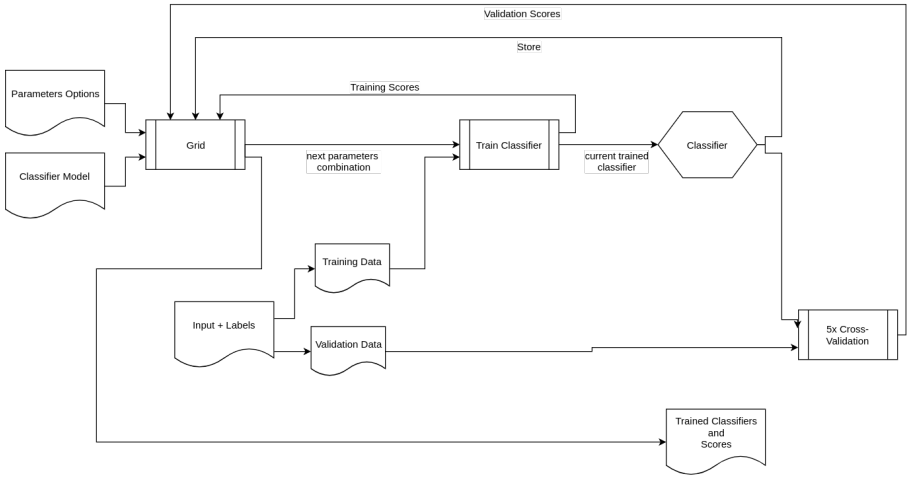

Figura 3.2: Diagrama explicando o processo de treinamento de um classificador com *GridSearch*.

3.6 Resultados

3.6.1 Volume de Dados

As quantidades de dados brutos adquiridas são:

- Primeira Coleta:
  - twitter.com: 36GB.
  - here.com:
    - Flow: 690GB;
    - Incidents: 3.5GB.
  - openweathermap.org: 6.3GB.
- Segunda Coleta:
  - twitter.com: 3.4GB.
  - here.com:
    - Flow: 235GB;
    - Incidents: 42GB.
  - openweathermap.org: 7.1GB.

Volume dos dados da segunda coleta após passar pelo processo de tratamento de dados explicado anteriormente:

– twitter.com: 405.7 MB, 1263289 *tweets*;

– here.com (Incidents): 10.7 MB, 15210 incidentes;

– openweather.org: 639 MB, 6080001 medições meteorológicas.

Os dados de fluidez de trânsito e de informações meteorológicas não foram utilizados em nenhum experimento até o momento.

3.6.2 Visualização

Para visualizar a distribuição dos incidentes no mapa foi implementado um *script* que transforma uma tabela, na qual cada evento ocupa uma linha, para outra, em que as linhas representam a longitude e as colunas a latitude, e cada célula contém a soma de eventos que se encaixam naquela longitude e latitude. Para isso é preciso fazer uma pivotação na tabela, já que em sua forma original tanto a informação de latitude e longitude são colunas. Fazer essa pivotação do *dataset* inteiro de uma só vez pode ser computacionalmente custoso dado seu tamanho. Por isso foi implementado um algoritmo que faz a pivotação iterativamente por pedaços. Além disso, é necessário estipular uma resolução para a tabela e truncar valores das coordenadas que caem no mesmo pixel. Depois de obtida a tabela de frequências, usa-se a biblioteca Seaborn [9] para transformá-la em um *heatmap*.

Para melhor localização visual, foi feito um *heatmap* por cima da imagem do mapa usando a biblioteca Folium [2]. Para a qual bastava mandar uma lista de coordenadas para produzir uma página em html com um mapa interativo, exemplificado na Figura 3.3.

Separando os incidentes por data e hora, foram feitos diversos mapas. Para visualizar a evolução dos incidentes foi criada uma animação, através de um *script* que fez captura de tela com cada mapa e os agrupou em um GIF (que pode ser visualizado em [8]).

Para melhorar a visualização no sentido de conseguir informar que tipo de evento aconteceu aonde, foi feito um outro mapa com a mesma biblioteca usando marcadores ao invés de um *heatmap*. A cor do marcador identifica a qual classe de incidentes ele pertence. É possível clicar no marcador e um balão com mais informações(horário exato e descrição) aparece acima dele. Como parâmetro do *script* que produz essas imagens, é possível especificar quais tipos de incidente aparecerão, como é mostrado na Figura 3.4 e na Figura 3.5, que apenas contém os incidentes do tipo congestionamento e acidente.

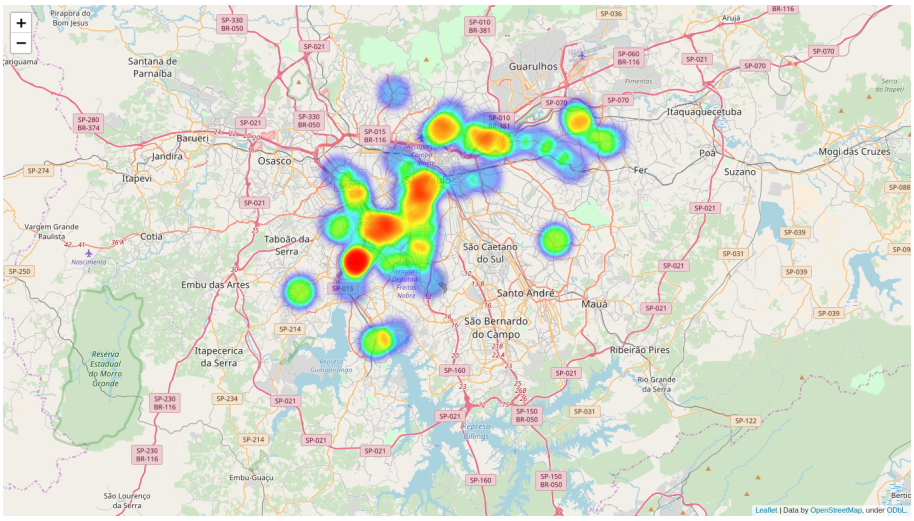

Figura 3.3: *Heatmap* dos incidentes que ocorreram as 7:43:00 no dia 12/06/2017 - Segunda-Feira.

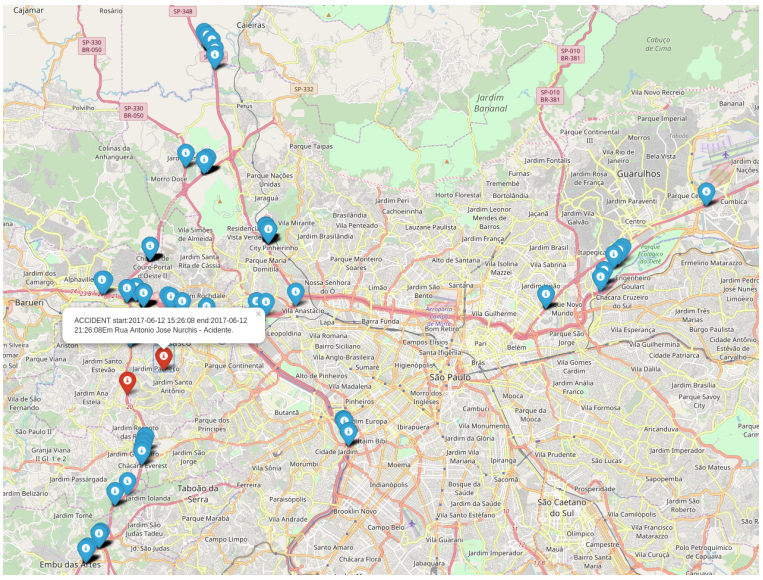

Figura 3.4: Mapa com marcadores dos Incidentes das classes Acidente e Congestionamento as 17:42:04 no dia 12/06/2017 - Segunda-Feira. Com um marcador de acidente selecionado.

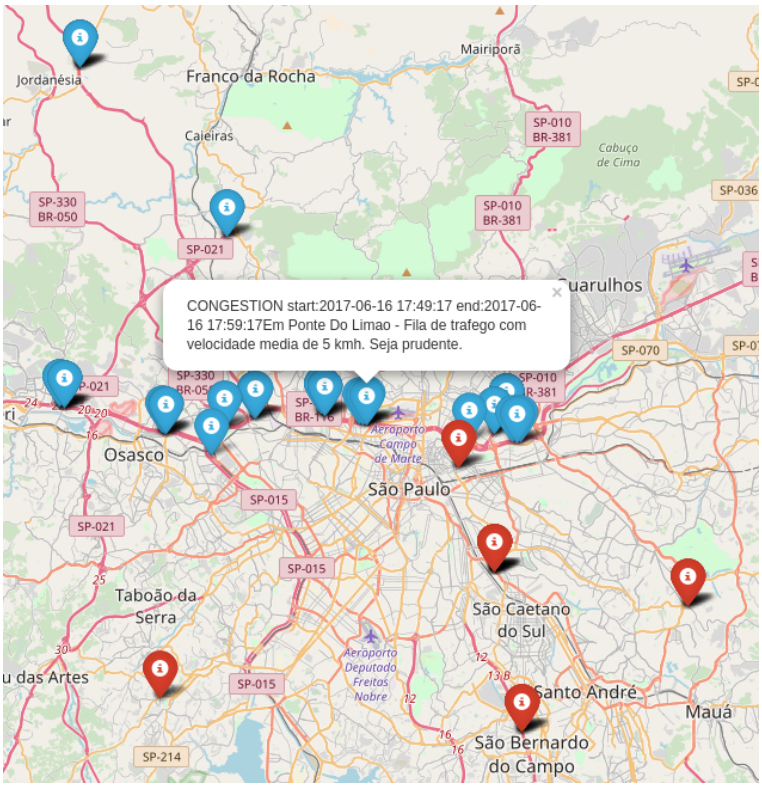

Figura 3.5: Mapa com marcadores dos Incidentes das classes Acidente e Congestionamento as 17:42:04 no dia 12/06/2017 - Segunda-Feira. Com um marcador de congestionamento selecionado.

Também foi empregado o uso de PCA e t-SNE para visualizar a expressividade das características do conjunto de dados de extração de endereço. Após a codificação com *word2vec* cada palavra se transformou em um vetor de tamanho 200. Para visualizar a primeira a distribuição das características da primeira palavra de cada postagem, foi inicialmente feita uma redução de 200 para 10 dimensões com PCA e depois de 10 para 2 com t-SNE. Não foi feita uma redução direta apenas com t-SNE por causa de seu custo computacional alto. O resultado do PCA apresentou 0.68 % da variância nas 10 características selecionadas. O resultado subsequente do t-SNE é mostrado na Figura 3.6. Observa-se uma certa separação entre as classes, que pode ser bem aproveitada pelos algoritmos de aprendizado de máquina.

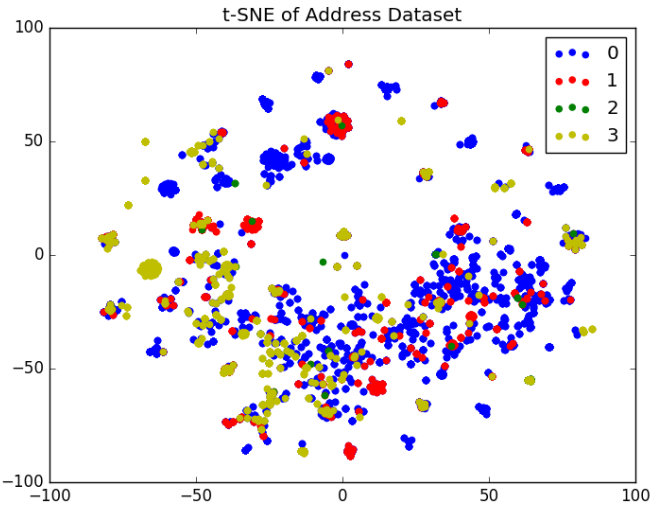

Figura 3.6: t-SNE das primeiras palavras do conjunto de treinamento para extração de endereços.

3.6.3 Codificação dos Dados

O uso *word2vec* obtido através do treinamento de um modelo de *skip-gram* melhorou significantemente a eficiência da classificação, como é apresentado na sessão a seguir. Deduz-se que isso se deve ao sucesso do método em representar palavras em forma de vetores numéricos de maneira que termos similares estejam próximos neste espaço vetorial. Verificou-se essa dedução consultando o modelo pelas palavras mais similares a um conjunto de termos escolhido. Foram feitos os seguintes testes, que resultam em pares de palavra e pontuação de similaridade:

- Busca pelas palavras mais similares a 'acidente' e 'carro':

```
[('ônibus', 0.7895407676696777), ('motorista', 0.7693743705749512),
('moto', 0.747714638710022), ('escada', 0.7393868565559387),
('porta', 0.7160496711730957), ('prédio', 0.7074517011642456),
('trem', 0.7013494372367859), ('elevador', 0.7010014653205872),
('apartamento', 0.6937683820724487), ('onibus', 0.6930466890335083)]
```

- Busca pelas palavras mais similares a 'acidente' e 'rodovia':

```
[('tanque', 0.7512314319610596), ('Ônibus', 0.7438685297966003),
('Sereias', 0.7406851053237915), ('chilena', 0.7393696904182434),
```

```
('espancado', 0.7386369705200195), ('vandalismo', 0.7373995780944824),
('retrovisor', 0.7364696860313416), ('abafa', 0.733789324760437),
('injustamente', 0.7329567670822144), ('óptero', 0.7308781147003174)]
```

- Busca pelas palavras mais similares a 'bom' e 'dia':

```
[('ótimo', 0.6664227843284607), ('ruim', 0.6302783489227295),
('fds', 0.5852736234664917), ('feliz', 0.5628766417503357),
('domingo', 0.5572034120559692), ('folga', 0.5560276508331299),
('belo', 0.5537296533584595), ('perfeito', 0.5507321357727051),
('solteiro', 0.53877854347229), ('feriado', 0.5330497026443481)]
```

Observando os resultados dos testes deduz-se que o modelo teve sucesso em capturar a similaridade de várias palavras com precisão. No entanto, também capturou similaridades que não fazem tanto sentido, como a similaridade entre ["Sereias"] e ["acidente", "rodovia"] já que não são palavras muito comum de serem encontradas juntas. Isso é uma evidência de que não houveram observações o bastante de alguma dessas palavras, o que indica que talvez a corpora não esteja compreensiva o bastante e que o modelo se beneficiaria de um conjunto com mais exemplos.

3.6.4 Fusão dos Dados

Na etapa de fusão de dados onde foi necessário incrementar o *dataset* dos *tweets* adicionando melhor geo-localização, o método de classificação apresentou resultados ruins ao ser treinado com os *tweets* submetidos apenas à primeira fase de codificação. Usando uma CNN foi possível obter taxa de acerto de 70%, mas observando a matriz de confusão (Tabela 3.3), a rede aprendia a acertar apenas o rótulo 0, que era o mais comum, e errava quase 100% das vezes nos outros. Foi então feita uma tentativa de balancear a frequência dos rótulos, retirando do conjunto de treinamento três quartos das entradas cujo rótulo é 0. Feito isso a taxa de acerto passou a ser cerca de 33%, e a matriz de confusão(Tabela 3.4) mostrou um equilíbrio menor. Foi testado também o conjunto de dados balanceado com uma SVM, usando *GridSearch* para otimizar os Hiper Parâmetros fazendo 5 validações cruzadas com cada combinação, o melhor classificador resultante obtinha uma taxa de acerto ao redor de 35%.

	0	1	2	3
0	4201	1	0	20
1	679	156	0	74
2	45	3	1	2
3	779	48	0	125

Tabela 3.3: Matriz de Confusão do primeiro experimento com a CNN para extração de endereços

	0	1	2	3
0	336	528	0	205
1	269	475	0	161
2	35	45	0	27
3	320	425	0	195

Tabela 3.4: Matriz de Confusão do segundo experimento com a CNN para extração de endereços

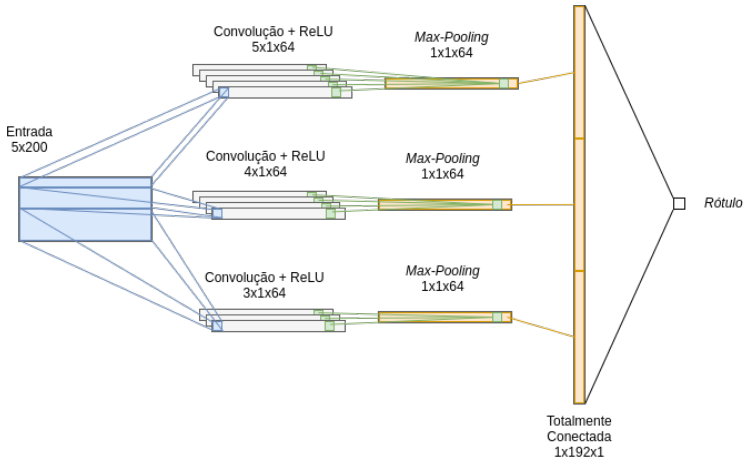

Figura 3.7: Detalhes da CNN usada no experimento de extração de endereços

Por sua vez, treinando a mesma rotina com dados submetido a todo o método de codificação (com *word2vec* agora) os resultados apresentados foram melhores, como mostrado na Tabela 3.5. Desta vez com uma taxa de acerto de cerca de 92%, e com uma sensibilidade (*recall*) melhor para todas as classes: 97%, 78% e 81%. Como foi observado no primeiro experimento que as entradas com rótulo 2 eram sempre confundidas com algum outro, este experimento foi feito considerando os rótulo 2 e 3 como iguais, já que na reconstrução do endereço isso pode ser tratado para que não afete a precisão da extração. A arquitetura da rede neural utilizada é mostrada na Figura 3.7

	0	1	2
0	4093	75	54
1	77	709	123
2	87	100	816

Tabela 3.5: Matriz de Confusão do terceiro experimento com a CNN para extração de endereços

Similarmente, ao usar o conjunto codificado, o SVM obteve resultados melhores em relação ao experimento anterior. Como indica a Tabela 3.6 e a Tabela 3.7, a performance da SVM linear e da SVM com núcleo RBF atingiram creca de 90% e 94% de taxa de acerto, respectivamente. E obtendo uma melhora significativano equilibrio entre as classes, apontado pela sensibilidade, como mostrado na Tabela 3.8. Os valores para os hiper-parâmetros selecionados pelo *GridSearch* foram todos 15, para o λ e para o C dos dois classificadores.

	0	1	2
0	4078	90	86
1	180	666	59
2	124	49	807

Tabela 3.6: Matriz de Confusão do segundo experimento com a SVM com núcleo linear para extração de endereços

	0	1	2
0	4173	36	45
1	66	807	32
2	63	39	878

Tabela 3.7: Matriz de Confusão do segundo experimento com a SVM com núcleo RBF para extração de endereços

Classe	SVM-Linear	SVM-RBF
0	96%	98%
1	74%	89%
2	82%	90%

Tabela 3.8: Sensibilidade de cada classe ao testar os classificadores SVM com núcleo linear e RBF.

No entanto, ao usar o classificador com o conjunto de dados de todos as postagens para extrair os endereços notou-se que este achava que havia endereço onde não deveria com muita frequência, contrariando a tabela de confusão obtida na validação com o conjunto de treinamento. Supõe-se que isso se deve ao fato do conjunto de postagens

usado no treinamento apenas conter textos cujo assunto é mobilidade urbana. Dessa forma, ao se deparar com textos tratando de outros assuntos o classificador está sujeito a se confundir consideravelmente. Uma solução para esse problema seria incluir entradas no conjunto de treinamento sobre outros assuntos, com rótulo 0.

O método de fusão por similaridade de texto mostrou-se extremamente custoso em relação a processamento e tempo de execução por necessitar de uma comparação palavra a palavra de todo o conjunto de dados. Por isso foi abortado após permanecer em execução por duas semanas, já que o método por classificação estava apresentando bons resultados.

3.6.5 Classificação de Incidentes

Como a etapa de fusão dos dados ainda não foi finalizada, não houve experimento de classificação dos incidentes. Esta tarefa é considerada de grande potencial e proposta como trabalho futuro.

Capítulo 4

Conclusão

O objetivo inicial do projeto era produzir um *dataset* a partir de coletas de dados de fontes heterogêneas, submetê-lo a um tratamento de filtragem e organização e com eles realizar análises usando aprendizado de máquina que se aproveitasse da fusão do conjunto. Conclui-se que o objetivo foi parcialmente cumprido, já que foi coletado um grande conjunto de dados de três fontes diferentes — *Twitter*, Trânsito e Meteorológico — nos quais foi feito um intensivo tratamento, e os quais também foram empregados no experimento de extração de endereços dos textos dos *tweets*, para o qual foram usadas algumas técnicas de aprendizado de máquina e processamento de linguagem natural. Além disso, foram produzidas visualizações que enriquecem a análise destes dados coletados.

Referências Bibliográficas

[1] Csv. https://www.loc.gov/preservation/digital/formats/fdd/fdd000323.shtml.

[2] Folium. https://python-visualization.github.io/folium/docs-master/.

[3] Here. https://developer.here.com/.

[4] Json. https://www.json.org/.

[5] Open weather api. https://openweathermap.org/api.

[6] Pandas. http://pandas.pydata.org/.

[7] Python geocoder. https://pypi.python.org/pypi/geocoder.

[8] Página contendo visualização dos dados em gif. https://linux.ime.usp.br/~luketis/mac0499/.

[9] Seaborn. https://seaborn.pydata.org/.

[10] Twitter api. https://dev.twitter.com/streaming/overview.

[11] Aho Alfred and Jeffrey Ullman. The theory of parsing, translation and compiling. 1972.

[12] Samuel Barbosa, Dan Cosley, Amit Sharma, and Roberto M. Cesar, Jr. Averaging gone wrong: Using time-aware analyses to better understand behavior. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 829–841, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.

[13] Andrey Bogomolov, Bruno Lepri, Jacopo Staiano, Nuria Oliver, Fabio Pianesi, and Alex Pentland. Once upon a crime: towards crime prediction from demographics and mobile data. In *Proceedings of the 16th international conference on multimodal interaction*, pages 427–434. ACM, 2014.

[14] Miao Chong, Ajith Abraham, and Marcin Paprzycki. Traffic accident analysis using machine learning paradigms. *Informatica*, 29(1), 2005.

[15] Justin Cranshaw, Raz Schwartz, Jason I Hong, and Norman Sadeh. The livehoods project: Utilizing social media to understand the dynamics of a city. In *International AAAI Conference on Weblogs and Social Media*, page 58, 2012.

[16] Laura Ferrari, Alberto Rosi, Marco Mamei, and Franco Zambonelli. Extracting urban patterns from location-based social networks. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, pages 9–16. ACM, 2011.

[17] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

[18] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[19] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] Rob Kitchin. The real-time city? big data and smart urbanism. *GeoJournal*, 79(1):1–14, 2014.

[21] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[22] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[23] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[25] Bei Pan, Yu Zheng, David Wilkie, and Cyrus Shahabi. Crowd sensing of traffic anomalies based on human mobility and social media. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 344–353. ACM, 2013.

[26] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges*

for NLP Frameworks, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[27] Cynthia Rudin, David Waltz, Roger N Anderson, Albert Boulanger, Ansaf Salleb-Aouissi, Maggie Chow, Haimonti Dutta, Philip N Gross, Bert Huang, Steve Ierome, et al. Machine learning for the new york city power grid. *IEEE transactions on pattern analysis and machine intelligence*, 34(2):328–345, 2012.

[28] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[29] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.11 0*, 2014.

[30] Shiliang Sun, Changshui Zhang, and Guoqiang Yu. A bayesian network approach to traffic flow forecasting. *IEEE Transactions on Intelligent Transportation Systems*, 7(1):124–132, 2006.

[31] Di Wang, Ahmad Al-Rubaie, John Davies, and Sandra Stinčić Clarke. Real time road traffic monitoring alert based on incremental learning from tweets. In *Evolving and Autonomous Learning Systems (EALS), 2014 IEEE Symposium on*, pages 50–57. IEEE, 2014.

[32] Paul John Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Doctoral Dissertation, Applied Mathematics, Harvard University, MA*, 1974.

Parte II

Parte Subjetiva

Capítulo 5

Considerações Pessoais

5.1 Desafios e Dificuldades

Com a primeira coleta de dados foi obtido um volume próximo dos 800GB e, por conta disso, muito tempo foi gasto tentando manejar essa quantidade colossal de dados em computadores do laboratório até que a decisão de fazer uma nova coleta com mais restrições fosse tomada. Simples operações em arquivos dessa magnitude demoram um tempo inábil mesmo em máquinas com boas configurações, o que implicava em longas esperas ansiosas monitorando o terminal (às vezes aberto por SSH no celular).

Mesmo vindo de forma estruturada (JSON), os dados minerados exigiram boa parte do tempo de todo o projeto só para serem devidamente tratados para compôr um *dataset* utilizável, e isso fez com que sobrasse pouco tempo para aplicar a análise encima destes conjuntos. Por conta disso, gostaria de continuar aprimorando essas análises.

Alguns assuntos da área de Aprendizado de Máquina, além de usarem ferramentas sofisticadas e, ás vezes por isso, complicadas, se encontram escritos de forma consideravelmente críptica na literatura, com muitos jargões e siglas. Ademais, me vi em situações em que havia implementado um modelo, testado-o, e este aparentava funcionar bem. Mas depois descobria que estava usando um modelo que não era adequado para a tarefa. Não obstante, esses desafios foram um grande incentivo para estudar bastante e exercitar a criatividade. Mas, mesmo com todo o esforço, vários impasses dos quais não conseguia sair foram encontrados. Por isso, principalmente nesses momentos, o direcionamento do Orientador foi de imensa importância para indicar a saída.

5.2 Disciplinas Importantes

- MAC0460 - Aprendizagem Computacional: Modelos, Algoritmos e Aplicações: Essencial para entender a teoria na qual se baseiam as técnicas de aprendizado de máquina;
- MAC0209 - Modelagem e Simulação:

Ensina as abordagens para se fazer experimentos e lidar com dados;

- MAE0221 - Probabilidade I, MAC0315 - Otimização Linear, MAC0300 - Métodos Numéricos da Álgebra Linear, MAT0122 - Álgebra Linear I e MAC0338 - Análise de Algoritmos:
  Uma característica marcante das técnicas de aprendizado de máquina é a quantidade de ferramentas matemáticas diferentes que são necessárias para entender sua teoria. Os conhecimentos de Probabilidade, Otimização, Algoritmos e Álgebra Linear obtidos nessas matérias são essenciais para entender desde o básico até os métodos mais complexos e elaborados;

- MAC0431 - Introdução à Computação Paralela e Distribuída:
  Nessa disciplina aprende-se como aproveitar vários computadores, vários processadores ou GPU's para fazer computação paralela e, consequentemente, otimizar o tempo de computação. Esse conhecimento é crucial para lidar com grandes quantidades de dados, já que, dependendo de sua ordem de magnitude, isso pode transformar uma computação impraticável (que demora anos, por exemplo) em algo razoável.

5.3 Considerações Finais

Com todas as dificuldades e barreiras, o TCC, assim como o curso em geral, me proporcionou uma ótima experiência por me obrigar a tentar fazer melhor, destrinchar problemas e, acima de tudo, perder o medo de tentar. Este trabalho, e as matérias também, exigem que de alguma forma e em algum momento você entre em contato com a ciência de ponta, seja através dos Professores ou da literatura, o que é algo que dá um pouco de apreensão, mas é muito gratificante.