Universidade Federal
do Rio de Janeiro
Escola Politécnica

IDENTIFICAÇÃO DE GÊNERO EM LETRAS MUSICAIS UTILIZANDO REDES PROFUNDAS E PYTEXT

Vinicius Almeida Alves

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola
Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Flávio Luis de Mello

Rio de Janeiro

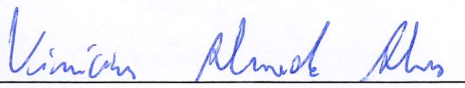
Setembro de 2019

IDENTIFICAÇÃO DE GÊNERO EM LETRAS MUSICAIS UTILIZANDO REDES PROFUNDAS E PyText

Vinicius Almeida Alves

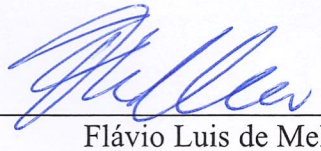
PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO

Autor:



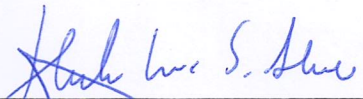
Vinicius Almeida Alves

Orientador:



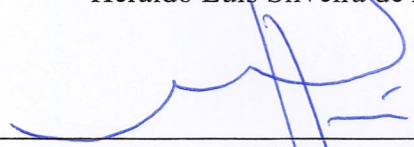
Flávio Luis de Mello, DSc.

Examinador:



Heraldo Luis Silveira de Almeida, DSc.

Examinador:



Manoel Villas Boas Junior, MSc.

Rio de Janeiro – RJ, Brasil

Setembro de 2019

Declaração de Autoria e de Direitos

Eu, *Vinicius Almeida Alves* CPF *132.593.687-10*, autor da monografia *Identificação de gênero em letras musicais utilizando redes profundas e PyText*, subscrevo para os devidos fins, as seguintes informações:

1. O autor declara que o trabalho apresentado na disciplina de Projeto de Graduação da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.
2. Excetuam-se do item 1. eventuais transcrições de texto, figuras, tabelas, conceitos e idéias, que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários, quando necessárias.
3. O autor permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação, a publicação do trabalho acadêmico em sua totalidade, ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ, ou aos seus representantes.
4. O autor pode, excepcionalmente, encaminhar à Comissão de Projeto de Graduação, a não divulgação do material, por um prazo máximo de 01 (um) ano, improrrogável, a contar da data de defesa, desde que o pedido seja justificado, e solicitado antecipadamente, por escrito, à Congregação da Escola Politécnica.
5. O autor declara, ainda, ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais, no que tange a cópia parcial, ou total, de obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro no art.184 e art.299, bem como na Lei 9.610.
6. O autor é o único responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados, não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/ indenização nesse sentido.
7. Por ser verdade, firmo a presente declaração.

Vinicius Almeida Alves

Vinicius Almeida Alves

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica – Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro – RJ CEP 21949-900

DEDICATÓRIA

Dedico este trabalho ao povo brasileiro que contribuiu de forma significativa à minha formação e estada nesta Universidade. Este projeto é uma pequena forma de retribuir o investimento e confiança em mim depositados.

AGRADECIMENTO

Inicialmente agradeço a todos os meus familiares que me ofereceram suporte e compreensão durante essa jornada que foi a graduação. Agradeço em especial à minha mãe que me motivou e me fez permanecer estudando mesmo em momentos de dificuldade, e também em especial à minha namorada que dividiu a carga pesada da universidade comigo, agradeço também ao meu pai, ao meu padrasto, à minha avó e às minhas tias e tios, por estarem presentes e terem ajudado ao longo do trajeto.

Gostaria de agradecer também à toda a comunidade da UFRJ, que fez com que eu me apaixonasse pela ciência e em especial por computação. Agradeço a todos os professores que me transmitiram o conhecimento que tenho hoje, e em especial ao professor Flávio Mello, por ter me cedido toda a atenção necessária para a elaboração deste trabalho. Agradeço também todo os servidores responsáveis pelo funcionamento da universidade. Agradeço também aos alunos que me ajudaram de diversas formas ao longo das disciplinas, pela amizade e dedicação.

Também preciso agradecer a todos os professores do meu ensino médio, que me deram a base para esta graduação, e também aos alunos e amigos que sempre estiveram presentes, mesmo após tanto tempo.

RESUMO

Desenvolvido há poucos meses pelo Facebook, o framework PyText tem por objetivo possibilitar o rápido desenvolvimento de novas aplicações usando técnicas recorrentes em problemas com textos, e ao mesmo tempo também possibilitar a exportação dessas aplicações para Caffe2, um ambiente de alta performance para produção. Assim, este projeto discorre sobre o uso do PyText em uma aplicação de classificação textual e o problema a ser resolvido será a identificação do gênero das letras musicais em Português Brasileiro presentes no portal Vagalume [1], utilizando as ferramentas disponíveis no framework.

Palavras-Chave: PyText, PyTorch, BiLSTM, CNN, NLP, processamento de linguagem natural, letras musicais.

ABSTRACT

Developed a few months ago by Facebook, the PyText framework aims to enable the rapid development of new applications using recurring text problem techniques, while also enabling the export of these applications to Caffe2, a high performance production environment.Thus, this project discusses the use of PyText in a textual classification application and the problem to be solved will be the identification of the genre of Brazilian Portuguese lyrics present in the Vagalume portal [1], using the tools available in the framework.

Key-words: PyText, PyTorch, BiLSTM, CNN, NLP, Natural Language Processing, Lyrics.

SIGLAS

BiLSTM – *Bidirectional Long Short-Term Memory*

CBOW – *Continuous Bag of Words*

CNN – *Convolutional Neural Network*

CPU – *Central Process Unit*

CSS – *Cascading Style Sheets*

CSV – *Comma-Separated Values*

CUDA – *Compute Unified Device Architecture*

Embrapa – Empresa Brasileira de Pesquisa Agropecuária

GAN – *Generative Adversarial Networks*

GPU – *Graphics Processing Unit*

HTML – *Hypertext Markup Language*

JSON – *JavaScript Object Notation*

LSTM – *Long Short-Term Memory*

MLP – *Multi Layer Perceptron*

NILC – Núcleo Interinstitucional de Linguística Computacional

NLP – *Natural Language Processing*

RAM – *Random-Access Memory*

ReLU – *Rectified linear units*

RNN – *Recurrent Neural Network*

SMOTE – *Synthetic Minority Over-sampling Technique*

Seq2Seq – *Sequence To Sequence*

TSV – *Tab-Separated Values*

UFRJ – Universidade Federal do Rio de Janeiro

USP – Universidade de São Paulo

UTF-8 – *8-bit Unicode Transformation Format*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 – Tema

O tema deste trabalho é o estudo do recente framework de processamento de linguagem natural PyText, desenvolvido pelo Facebook com base em PyTorch. Ele vem com a premissa de otimizar o desenvolvimento de modelos preditivos baseados em texto, alinhando rápida prototipação e alto desempenho em produção.

Neste sentido, o problema abordado será a identificação de gêneros de letras musicais usando o PyText.

1.2 – Delimitação

O objeto de estudo deste trabalho é a classificação de texto livre - representado por letras musicais - usando os modelos e técnicas disponíveis na ferramenta PyText. Não são abordadas outras técnicas presentes no framework que não aquelas direcionadas para a tarefa de classificação.

O conjunto de dados de letras musicais utilizados para classificação será restringido à linguagem português do Brasil.

1.3 – Justificativa

Com o grande avanço da área de redes neurais nos últimos anos, a performance de modelos preditivos baseados em métodos de processamento de linguagem natural tem se expandido, possibilitando o desenvolvimento de diversas aplicações baseadas em análise de texto, tais como: identificação de sentimento em sentenças, tradução de texto, reconhecimento de entidades em texto, reconhecimento de fala, descrição de imagens, etc [2].

Neste cenário, foram criadas diversas ferramentas que auxiliam o desenvolvimento destes modelos, sendo que estas podem ser segmentadas em níveis de abstração distintos, priorizando rápida prototipação ou alta performance. O PyText entra com a proposta de atender ambas as necessidades, trazendo rápido desenvolvimento e também alto desempenho em produção.

Entretanto, a comunidade do novo framework ainda é bem restrita, a documentação também não é completa. Deste modo, o presente trabalho pode auxiliar aos novos usuários, trazendo uma descrição clara da ferramenta através do estudo de caso proposto.

1.4 – Objetivos

O objetivo geral é apresentar as funcionalidades da ferramenta PyText através do estudo de caso da identificação de gênero de letras musicais. Desta forma, tem-se como objetivos específicos: (1) preparar um conjunto de dados para possibilitar testes dos modelos preditivos; (2) apresentar o processo de uso do software PyText, da instalação à execução; (3) demonstrar de forma macroscópica a arquitetura interna do PyText; (4) detalhar os modelos preditivos disponíveis para o caso em análise.

1.5 – Metodologia

A pesquisa foi dividida em cinco subáreas: (1) obtenção dos dados; (2) preparação do *dataset* para classificação; (3) estudo da ferramenta PyText; (4) estudo dos modelos preditivos disponíveis na ferramenta para a tarefa especificada; (5) desenvolvimento, execução e avaliação dos modelos selecionados para o conjunto de dados selecionado.

Inicialmente, foi realizada uma etapa de pesquisa dentre as fontes de dados de texto disponíveis em Português que se adequassem ao problema de classificação, e, desse modo, foi selecionado um conjunto de dados de letras musicais presentes no site Vagalume [1]. Como este conjunto estava incompleto, contando apenas com seis gêneros, foi aprimorado o *script* responsável pela obtenção dos dados, de modo a obter letras de todos os gêneros, também corrigindo erros que o *script* não tratava.

Em seguida, foi realizado uma higienização das letras, filtrando as de outros idiomas e corrigindo as letras com conteúdo extra, como tags e notas de rodapé. Também

foi realizado estudo e mapeamento das letras para o gênero adequado, já que uma letra pode estar previamente mapeada em mais de um gênero.

Em posse dos dados prontos para classificação, foi realizado um estudo do framework PyText e das redes neurais disponíveis para classificação de texto, inclusive de formas de lidar com o grande desbalanceamento entre as classes do *dataset*.

Logo após foi realizada a instalação da ferramenta, e com isso a execução e avaliação dos modelos disponíveis para o conjunto de dados produzido.

1.6 – Descrição

O capítulo 2 apresenta toda a pesquisa que foi feita para o presente trabalho. Inicialmente discorre sobre o problema de classificação de texto, em seguida são comparadas duas ferramentas utilizadas para a confecção de novas arquiteturas de redes neurais, após isto toda a problemática do PyText é exposta, incluindo seções sobre instalação, configuração, execução, arquitetura interna, etc. A seguir são abordados pontos em relação à classificação de textos, como formas vetoriais de representação do texto e técnicas para tratar *datasets* desbalanceados. E por fim são demonstradas duas arquiteturas de redes neurais utilizadas para classificação de texto disponíveis no PyText.

A etapa de elaboração do *dataset* utilizado para classificação é apresentada no capítulo 3. Inicialmente há uma etapa de seleção entre os conjuntos de dados em Português encontrados, na qual o conjunto de letras musicais é selecionado, entretanto ele não estava completo, o que leva à próxima seção na qual houve um aperfeiçoamento no *script* que gerou este conjunto. A seguir é realizada uma análise exploratória dos dados, a qual culmina no problema de definir o gênero adequado para cada letra musical, já que uma mesma letra pode ter vários gêneros associados.

No capítulo 4 é descrito o uso do PyText para o *dataset* gerado na seção anterior. Neste trabalho é usada a versão 0.2.0 da ferramenta. São desenvolvidos dois modelos preditivos para o problema, em seguida são comparados os resultados de ambos, de forma geral e também por classe.

Por fim, o capítulo 5 evidencia as conclusões do trabalho, e também expõe sugestões para próximos trabalhos no mesmo tema.

Todo o código produzido no capítulo 4 e também os arquivos de configurações e resultados do capítulo 5 estão disponíveis no github associado, em [3].

Capítulo 2

Fundamentação teórica

2.1 – Classificação de documentos

Nos últimos anos vem ocorrendo uma grande evolução no campo de redes neurais com a construção de novas arquiteturas de redes e também com a elaboração de grandes frameworks que possibilitam a rápida prototipação e produção industrial das mesmas. Tal evolução também se deu pela grande quantidade de dados que a sociedade tem produzido atualmente, numa escala muito maior que outrora. Outro fator influenciador foi o crescimento da capacidade de processamento dos computadores, chegando a ser possível processar redes com centenas de camadas.

Este movimento, por sua vez, provocou a alta das áreas de aprendizado de máquina, incluindo o segmento de processamento de linguagem natural, que aborda diversos problemas relacionados à linguagem, como: classificação de documentos, reconhecimento de entidades, resumo automático de texto, etc.

Deste modo, a classificação de documentos é um problema muito pesquisado, com diversas aplicações mercadológicas, que consiste em identificar um determinado atributo - que pode ser o tema, o autor, o sentimento que o autor manifestava durante a escrita, a veracidade do texto - em diferentes textos.

Sendo assim, no decorrer do tempo, foram desenvolvidas diversas técnicas de análise e também arquiteturas de redes neurais com objetivo de auxiliar neste processo de classificação.

2.2 – PyTorch e TensorFlow

A construção e rápida prototipação de redes neurais é crucial para o desenvolvimento de novos modelos preditivos. Há bibliotecas de uso simples e rápido, tal como o scikit-learn [4], construída na linguagem Python e com dezenas de modelos preditivos à disposição. Entretanto, tal biblioteca não atende diversos requisitos habituais do desenvolvimento de modelos de classificação baseados em redes neurais, tais como:

- Flexibilidade para produção de novas arquiteturas e variantes das atuais;
- Alto grau de paralelismo utilizando até GPUs se especificado;
- Possibilidade de alterar o modelo em tempo de execução.

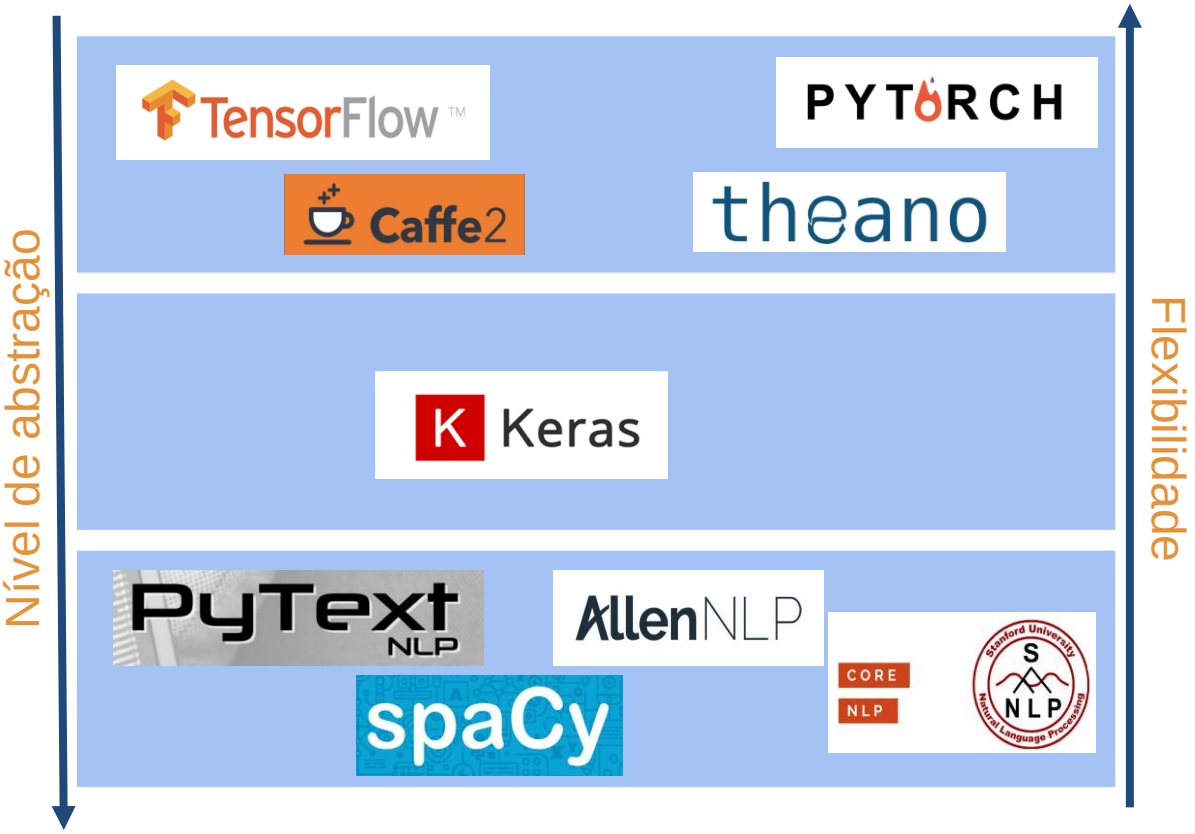


Figura 2.1 – Segmentação entre ferramentas NLP em nível de abstração e flexibilidade.

Para atender estes requisitos surgiram diversos novos frameworks no mercado, que podem ser separados em diferentes níveis de abstração e flexibilidade, assim como pode ser visto na figura 2.1. Dentre os primeiros que surgiram os que mais se destacaram foram PyTorch [5], uma iniciativa do Facebook, e TensorFlow [6], uma iniciativa do Google. Eles permitem descrever novas arquiteturas de redes neurais via linguagem de programação, ou em alguns caso, em arquivos de texto estruturado, tais como JSONs. Também permitem grande otimização e paralelismo na execução, algo necessário quando se pretende utilizar grandes quantidades de dados em redes profundas.

Há algumas diferenças bem relevantes entre estes frameworks, tais como [7, 8, 9]:

- TensorFlow é baseado em grafos estáticos, enquanto PyTorch é baseado em dinâmicos, ou seja, com o primeiro é necessário descrever os modelos como grafos, após descrito o grafo basta executá-lo. Já o PyTorch é imperativo, ou seja,

é necessário descrever o passo-a-passo para a execução, o que torna o desenvolvimento de modelos mais flexível e facilita a correção de erros;

- PyTorch utiliza o próprio debugger do Python, o que facilita o processo de identificação de erros, já o TensorFlow utiliza um debugger externo;
- A curva de aprendizado do PyTorch é menor que a do TensorFlow. Uma vez que o TensorFlow requer que os usuários aprendam novos conceitos sobre sua estrutura interna;
- Os modelos gerados com TensorFlow podem ser totalmente serializados. Isto é, uma vez que o grafo esteja construído é possível salvá-lo em um formato padronizado e até exportar para outras linguagens suportadas. Isto já não é possível com PyTorch;
- Para servir os modelos em produção, TensorFlow conta com a ferramenta TensorFlow Serving, que é um sistema de alta performance desenhado para servir modelos. Já PyTorch possui integração com o microframework Flask;
- TensorFlow também permite o *hot swap* de modelos, ou seja, atualizar um modelo em produção sem ser necessário pausar o serviço;
- A documentação de ambos possui boa qualidade. Contando com explicações detalhadas, exemplos e tutoriais;
- Para aumentar o grau de paralelismo utilizando até GPUs no PyTorch é necessária determinada configuração, já o TensorFlow permite o paralelismo mais direto, de forma transparente.

Um ponto negativo destes dois frameworks é que seu uso é muito geral, tornando trabalhoso descrever algum modelo já conhecido de forma simplificada. Sendo assim, foram desenvolvidos outros frameworks para diferentes tarefas que utilizam estes como base.

2.3 – PyText

2.3.1 – Introdução

Entre os frameworks desenvolvidos com objetivo de atacar problemas na área de linguagem natural se destacam: CoreNLP, AllenNLP, Spacy 2.0 e PyText [10]. Todos possuem modelos e técnicas de processamento para redes neurais, incluindo tanto as ferramentas clássicas, quanto as desenvolvidas nos últimos anos. Cada um dos frameworks possuem características próprias, alguns são mais performáticos e outros mais amigáveis ao usuário.

O PyText foi desenvolvido pelo Facebook utilizando PyTorch como base, e disponibilizado no final do ano de 2018. Segundo Ahmed, ele tem por meta atuar como um framework que alia alta performance, as principais arquiteturas de *deep learning* para texto e rápida prototipação. Esta combinação ainda não é presente em nenhum outro framework da área, tornando o software promissor [10].

Com PyText, o usuário pode facilmente prototipar com modelos já disponibilizados ou criar hierarquias personalizando módulos internos dos modelos, de forma a adequar o modelo aos seus dados e ao seu problema. Também é possível criar novos modelos alterando apenas as partes necessárias, já que é totalmente baseado em módulos.

Para executar algum modelo preditivo é necessário descrever um arquivo JSON com as configurações dele, contando com descrição das entradas e saídas, entre outros parâmetros. Após isto, basta chamar o executável do PyText pela linha de comando e dependendo das opções selecionadas, ele irá treinar uma rede e no final gerar um arquivo com o modelo pronto em si. Com este arquivo é possível utilizar o modelo desenvolvido em produção através do framework Flask. Todavia, uma das grandes vantagens da ferramenta é a possibilidade de exportar modelos para Caffe2, possibilitando servir modelos para aplicações intensivas com alto desempenho.

Devido ao pouco tempo de desenvolvimento o PyText possui apenas algumas arquiteturas para diferentes tarefas [11]:

- Para classificação de texto existem duas arquiteturas, uma baseado em uma rede BiLSTM com atenção e outra baseada em uma rede CNN;
- Para identificação de entidades há uma rede do tipo BiLSTM;

- Também há uma arquitetura chamada *Joint intent-slot*, que serve para identificação de intenção e filtragem de *slots* de interesse em um texto, este por sua vez também é baseado em uma rede BiLSTM;

Olhando pelo código-fonte é possível ver que existem outras arquiteturas, todavia não estão bem descritas na documentação. Arquiteturas do tipo Seq2Seq, que levam uma sequência de texto para outra sequência, como em traduções, ainda não estão disponíveis na ferramenta.

2.3.2 – Instalação e execução

Para instalar a ferramenta é necessário possuir uma máquina com Python 3.6 ou superior. É recomendável utilizar Ubuntu 18 ou superior, pois foram realizados testes com outras distribuições Linux (Mint, Fedora) e estas apresentaram problemas na instalação. Também não foi possível executar no Windows, pois a ferramenta ainda não é totalmente compatível. Para rodar o framework usando uma GPU com CUDA, é necessário possuir a placa em questão e instalar os drivers específicos – sendo estes, por sua vez, difíceis de serem instalados nas distribuições Linux. Para futuros trabalhos, seria interessante realizar testes no Mac OS X.

Para instalar os drivers da GPU CUDA o caminho mais simples é executar a série de comandos no Ubuntu:

```
sudo add-apt-repository ppa:graphics-drivers/ppa
sudo apt update
sudo apt-get install ubuntu-drivers-common
sudo ubuntu-drivers autoinstall
sudo shutdown -r now # reboot
sudo apt install nvidia-cuda-toolkit gcc-6
```

Figura 2.2 – Sequência de comandos para se instalar drivers CUDA [11].

Mesmo com a sequência acima ainda podem ocorrer problemas. De todo modo, não é necessário possuir GPU CUDA para rodar o PyText, sem os drivers ele funciona utilizando apenas a CPU.

A seguir, basta instalar diretamente a ferramenta, executando os comandos abaixo no Ubuntu. Caso o PyTorch não esteja instalado, ele será instalado pelo gerenciador de pacotes do Python.

```
sudo apt-get install protobuf-compiler libprotoc-dev
pip install pytext-nlp
pip install tensorflow tensorboard
```

Figura 2.3 – Sequência de comandos para se instalar o PyText e o TensorBoard [11].

Uma vez que o PyText esteja instalado, estará disponível o executável dele na linha de comando. Para executar qualquer modelo é necessário chamar este executável passando o tipo de execução e um arquivo JSON de configuração como argumento, seguem exemplos abaixo.

```
# treina um modelo e salva o resultado em arquivo
pytext train < config.json

# testa um modelo salvo
pytext test < config.json

# exporta um modelo salvo para Caffe2
pytext export < config.json
```

Figura 2.4 – Exemplos de comandos para execução do PyText [11].

Ao executar o PyText no modo treino ele irá logar no terminal por padrão o resultado de cada *epoch* de treino, incluindo métricas de: precisão, cobertura e acurácia, assim como pode ser visto na figura 2.5.

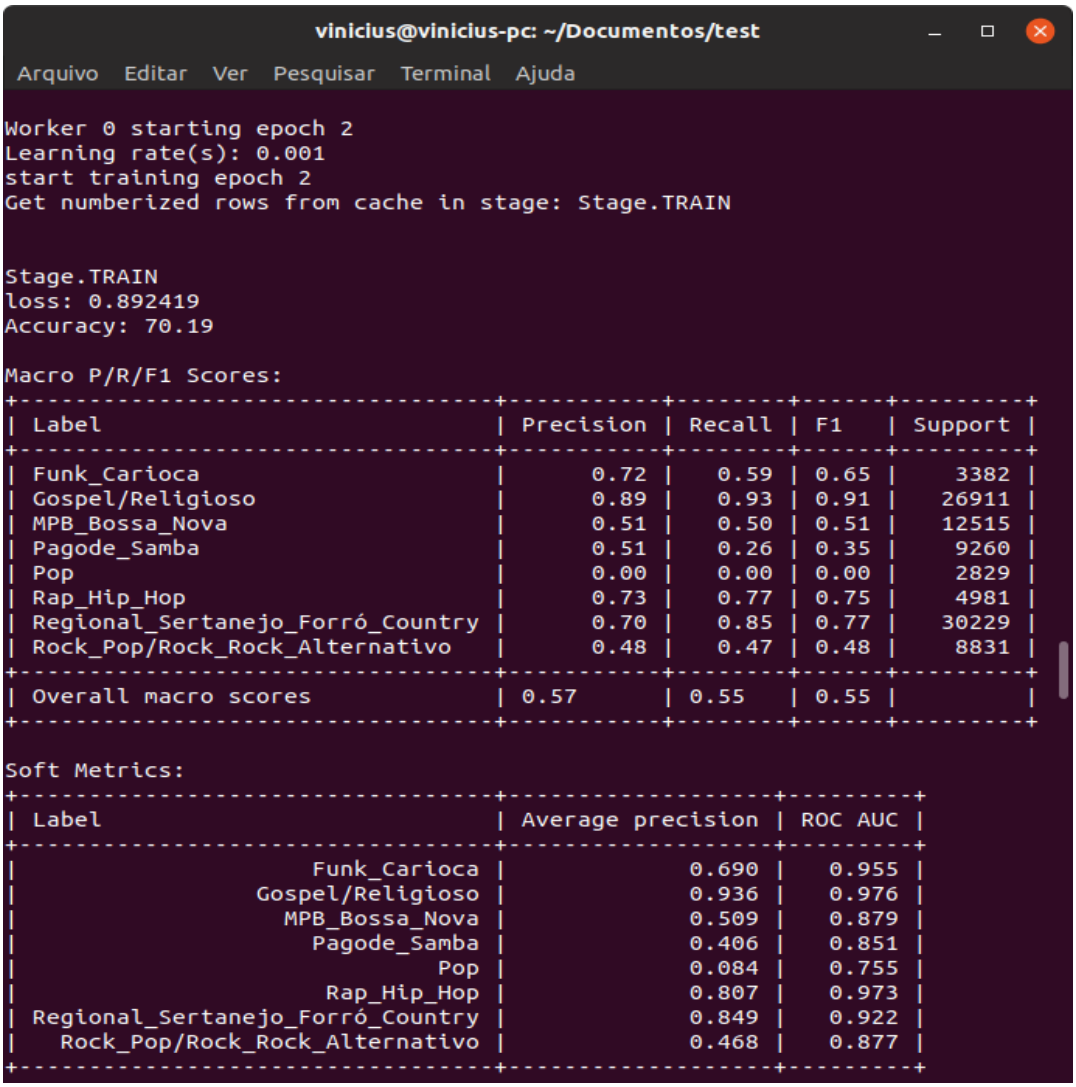


Figura 2.5 – Exemplo de saída do PyText durante o treino.

Além de acompanhar pelo terminal, também é possível utilizar o TensorBoard, uma ferramenta do TensorFlow usada para acompanhar e comparar o progresso do aprendizado de diferentes modelos. Com ele é possível visualizar graficamente a evolução da acurácia e custo ao longo das *epochs*. Um exemplo de uso do TensorBoard com o PyText está disponível na figura 2.6. O TensorBoard pode ser usado em conjunto com qualquer framework, desde que este ofereça suporte.

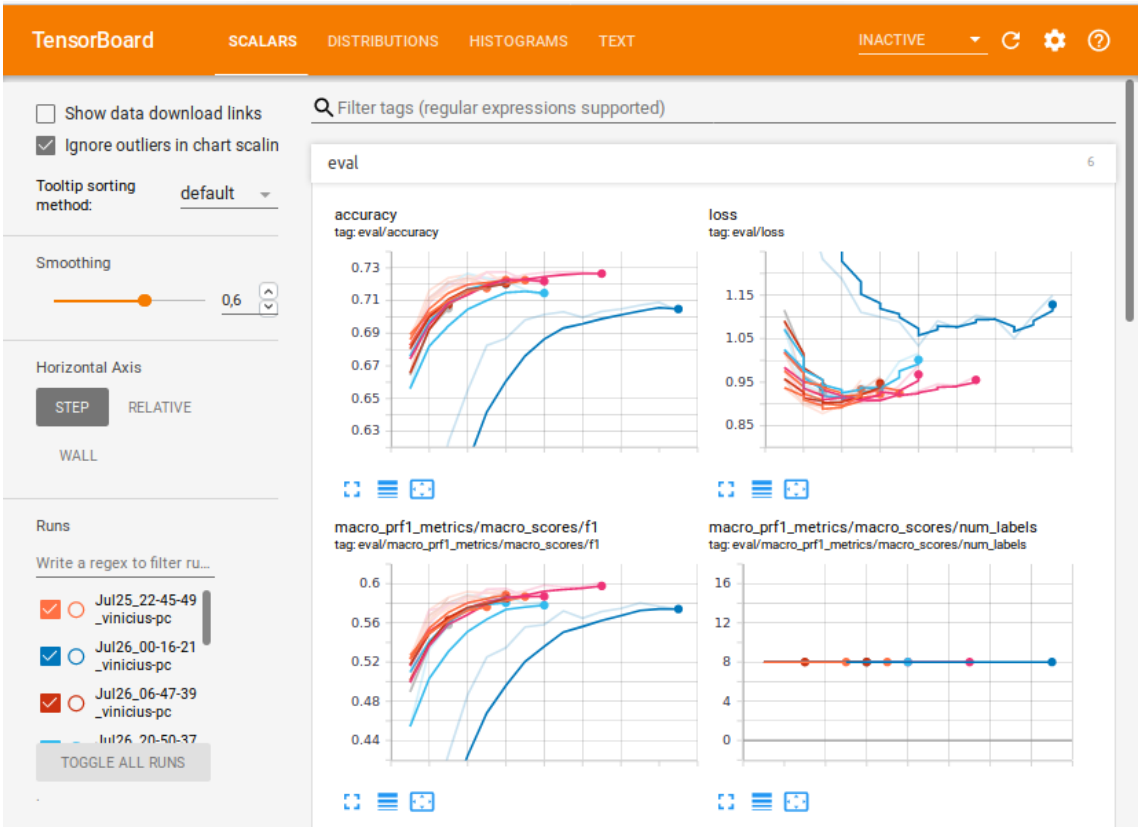


Figura 2.6 – Exemplo de uso do TensorBoard.

2.3.3 – Arquitetura

O PyText é formado, essencialmente, por módulos responsáveis por diferentes etapas dos modelos preditivos e cada módulo é definido por uma classe Python. Para diferentes módulos, podem existir diversas opções de classes a serem utilizadas e também hierarquias.

O módulo Task é o responsável pela interface externa: entrada, saída, exportação de modelos, e configurações gerais de treinamento. Dentro dele estão outros módulos, como:

- TSVDataSource: módulo padrão para carregar arquivos TSV, este por sua vez é o formato padrão utilizado pela ferramenta. Na configuração do Task o usuário deve especificar os caminhos para arquivos TSV de treino, avaliação e teste. Caso o usuário opte por outro formato de entrada não existente deverá criar um novo DataSource;
- Batcher: responsável por agrupar os dados de treino em lotes que serão repassados para as redes neurais. Cada grupo é formado de acordo com o hiperparâmetro delimitado no JSON de entrada, os grupos são gerados de

forma sequencial ao longo do *dataset*, todavia também há outra versão do módulo com geração randômica;

- Trainer: nele são configuradas as opções mais gerais de treinamento, como número de *epochs*, *random_seed*, opções de visualização dos resultados no terminal, etc. Ele também é responsável por atualizar os pesos utilizados nas redes neurais a cada *batch*;
- MetricReporter: tem por objetivo logar as métricas de interesse no console e permitir aos usuários escolherem quais métricas desejam logar;
- Exporter: responsável pela exportação do modelo, geralmente em Caffe2.

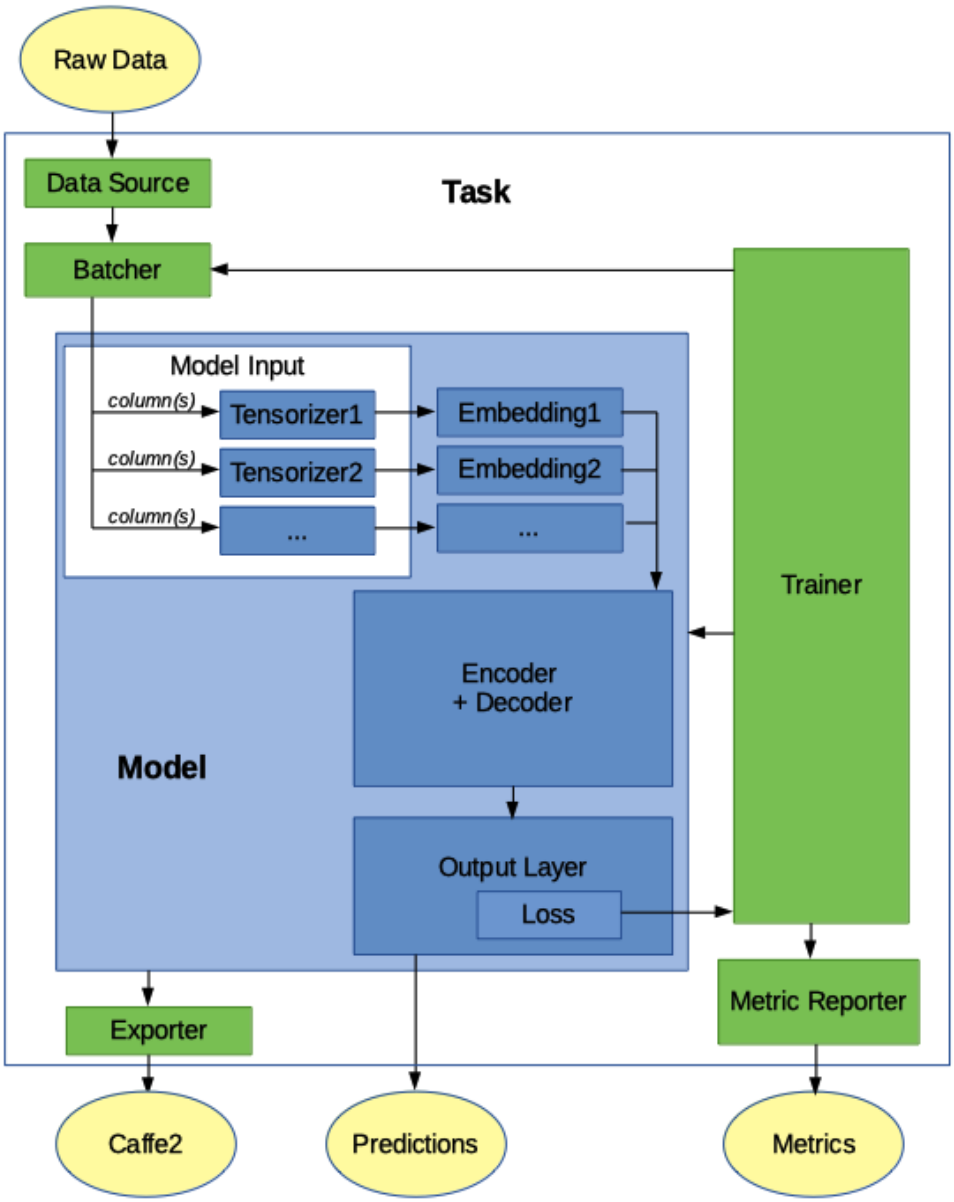


Figura 2.7 – Diagrama da macroarquitetura do PyText. [11]

Dentro do módulo Task há o módulo Model, que é o local onde fato acontece o pré-processamento das entradas e treinamento das redes neurais. Ele pode ser subdividido em:

- ModelInput: age em conjunto à família de módulos Tensorizer, que tem por objetivo mapear cada linha de texto em um Tensor. Diferentes tipos de dados necessitam de diferentes Tensorizers. Cada objeto Tensorizer inicialmente cria um vocabulário, que é um dicionário mapeando todas as palavras encontradas em determinada coluna no conjunto de treino em um índice. Para cada linha da entrada haverá um tensor, ou uma tupla de tensores de saída, que serão essencialmente vetores contendo os índices das palavras naquele texto mapeadas no dicionário;
- Embedding: converte os tensores criados na etapa anterior em *embeddings*, podendo ser do tipo WordEmbedding ou CharacterEmbedding;
- Representation: também chamada de encoder, é a camada onde se situa o núcleo da rede neural, que poderia ser uma rede CNN, LSTM, etc. Esta camada recebe os *embeddings* e também carrega as matrizes de pesos associadas à cada possível representação de rede;
- Decoder: Geralmente uma camada MLP utilizada para decodificar as classes de saída. Cada classe de saída possuíra um neurônio de saída respectivo;
- OutputLayer: Identifica a classe mais provável para cada entrada através do Decoder e com isso realiza o mapeamento do identificador desta classe nos labels que foram usados anteriormente em treinamento. Também é responsável por calcular a função de custo, *loss*, e repassar o resultado para o módulo Trainer.

2.3.4 – Configuração

Como explorado na seção 2.3.2, é necessário utilizar um arquivo JSON de configuração junto ao PyText. Este arquivo tem uma estrutura fixa definida pelos módulos e submódulos a serem utilizados no modelo preditivo. O PyText oferece na sua

documentação, e também via linha de comando, exemplos de arquivos de configuração padrões para diferentes tarefas.

Cada objeto de dados do JSON, ou seja, cada conjunto de par-valor delimitado entre chaves na estrutura, representa um objeto, sendo os conjuntos de par-valor seus atributos.

Também é possível inferir a estrutura das classes a partir dos objetos de dados do JSON, por exemplo na figura 2.8 pode-se ver que DocumentClassificationTask possui alguns objetos internos, como: *data*, *trainer* e *model*. Já o DocModel possui *embedding*, *representation* e *output layer*. Também podemos visualizar o módulo TSVDataSource, que por sua vez é uma classe filha da classe base DataSource.


```
{
   "task":{
      "DocumentClassificationTask":{
         "data":{
            "Data":{
               "source":{
                  "TSVDataSource":{
                     "train_filename":"data/train_data.tsv",
                     "test_filename":"data/test_data.tsv",
                     "eval_filename":"data/eval_data.tsv"
                  }
               },
               "batcher":{
                  "PoolingBatcher":{
                     "train_batch_size":16
                  }
               }
            }
         },
         "trainer":{
            "epochs":10,
            "early_stop_after":1,
            "optimizer":{
               "Adam":{
                  "lr":0.001,
                  "weight_decay":1e-05
               }
            }
         },
         "model":{
            "DocModel":{},
            "embedding":{
               "embed_dim":100,
               "embedding_init_strategy":"random",
```

```
            "pretrained_embeddings_path":"glove_s300.txt"
         },
         "representation":{
            "BiLSTMDocAttention":{
               "dropout":0.4,
               "lstm":{
                  "dropout":0.4,
                  "lstm_dim":32,
                  "num_layers":1,
                  "bidirectional":true
               },
               "pooling":{
                  "SelfAttention":{
                     "attn_dimension":64,
                     "dropout":0.4
                  }
               }
            }
         },
         "output_layer":{
            "loss":{
               "CrossEntropyLoss":{}
            }
         }
      }
   }
}
```

Figura 2.8 – Exemplo de arquivo JSON de configuração para o PyText.

No JSON há campos opcionais e obrigatórios para cada objeto da arquitetura usada, por exemplo, em TSVDataSource constam os caminhos de entrada de dados como campos obrigatórios. Em geral, a maior parte dos campos já possui atributos padrões e o que geralmente muda de um modelo para outro são os submódulos, para classificação de texto por exemplo podem ser usadas diversas arquiteturas internas de rede neural (*representations*), também podem ser usadas várias medidas para calcular a função de custo (*loss*).

Caso o usuário ao escrever um novo módulo, ou seja uma classe Python, deverá registrar esse componente através do executável PyText. Ao criar um novo *representation*, por exemplo, o usuário poderá deixar atributos com valores pré-definidos ou não, estes por sua vez, após o registro do módulo, serão passíveis de serem

configurados via JSON. Dependendo do módulo a ser escrito pode ser necessário que o usuário utilize ferramentas do PyTorch.

2.3.5 – Exportação para Caffe2

Assim como TensorFlow, Caffe2 é um ambiente para a produção de modelos preditivos baseados em redes neurais através de grafos estáticos. Foi construído pelo Facebook utilizando apenas C++, seu principal objetivo é alta perforamance em produção e também baixa necessidade de processamento em *mobile*.

O PyText utiliza o ONNX - Open Neural Network Exchange Format - para descrever os modelos produzidos e também para possibilitar a migração da aplicação para um grafo estático Caffe2. O modelo descrito em Python tem problemas ocasionados pela linguagem, como: (1) Perfomance menor quando comparado à linguagens compiladas; (2) Sem suporte a *multithreading* utilizando *kernel level threads*, algo que limita o paralelismo para o ambiente de processos.

Na tabela 2.1 é possível verificar um comparativo da latência do modelo JointBiLSTM exportado pelo PyText para PyTorch e para Caffe2. É visível que a implementação em Caffe2 obteve uma latência em média 50% menor.

Tabela 2.1 – Comparação de latência (milisegundos) no modelo JointBiLSTM exportado para PyTorch e Caffe2 [10].

Modelo	**Implementação**	**P50**	**P90**	**P99**
JointBiLSTM	PyTorch	34.08	47.23	64.94
	Caffe2	19.65	24.69	30.21

2.3.6 – Documentação e comunidade

O PyText possui uma documentação abrangente sobre todos os seus módulos [11]. Essa documentação também conta com alguns tutoriais, passo-a-passo para instalação, etc. Entretanto, os tutoriais são poucos e com pouca profundidade. Neste trabalho, por muitas vezes, foi necessário mergulhar no código-fonte para entender melhor determinados pontos. Embora exista a descrição da existência de todos os módulos na

documentação, com frequência não existe informação sobre um parâmetro ou outro para determinados módulos.

A comunidade de usuários pode ser resumida ao grupo oficial PyText do Facebook [12]. Este grupo conta com alguns dos desenvolvedores da ferramenta e cerca de 600 membros. Ou seja, a comunidade ainda é bem pequena, algo esperado para o início de um projeto, configurando, assim, uma barreira para o uso da ferramenta.

2.4 – Formas de representação vetorial de texto

Para realizar qualquer tipo de aprendizado sobre dados do tipo texto, é necessário de antemão definir alguma técnica para mapear o texto em vetores numéricos. Existem alguns métodos para tratar este problema e todos eles iniciam seu processamento tokenizando o texto, ou seja, quebrando o texto em fragmentos. O modo mais simples de realizar essa tokenização é quebrar o texto a cada espaço em branco, com isso cada texto se tornará um vetor de tokens. Em posse deste vetor os métodos criam um vocabulário, que contém todos os possíveis tokens ao longo do *dataset*.

Em posse do vocabulário e do vetor de tokens para cada texto, é necessário verificar métodos para transformar este vetor de tokens em um vetor numérico. Uma das formas mais antigas e mais conhecidas de fazer isso é o *bag-of-words*. Neste método cada texto será mapeado em um vetor numérico de tamanho fixo igual ao do vocabulário onde cada posição representa a quantidade de ocorrências daquele token no vetor de tokens daquele texto.

Por exemplo, considere os textos abaixo:

1. Maria gosta de laranja
2. João e Maria gostam de frutas
3. João não gosta de pêssego

Transformando-os em vetores de tokens teríamos a configuração exibida na figura 2.9, na mesma figura também há um possível vocabulário para o conjunto.

```
# vetores de tokens
1. ['Maria', 'gosta', 'de', 'laranja']
2. ['João', 'e', 'Maria', 'gostam', 'de', 'frutas']
3. ['João', 'não', 'gosta', 'de', 'pêssego']
```

```
# possível vocabulário
vocab = {'Maria': 0, 'gosta': 1, 'de': 2, 'laranja': 3, \
'João': 4, 'e': 5, 'gostam': 6, 'frutas': 7, 'não': 8,
'pêssego': 9}
```

Figura 2.9 – Exemplo de representação vetor de tokens e vocabulário.

Cada chave do vocabulário é um token encontrado ao longo do *dataset* e cada valor da chave representa um índice para aquele token. Em posse do vocabulário e dos vetores de tokens podemos calcular a representação *bag-of-words* daquele conjunto. Como há 10 valores no vocabulário, cada vetor resultante possuirá 10 posições, que indica o número de vezes que aquele termo do vocabulário apareceu no vetor de tokens, assim como pode ser visto na figura 2.10.

```
1. [1, 1, 1, 1, 0, 0, 0, 0, 0, 0]
2. [1, 0, 1, 0, 1, 1, 1, 1, 0, 0]
3. [0, 1, 1, 0, 1, 0, 0, 0, 1, 1]
```

Figura 2.10 – Exemplo de representação *bag-of-words*.

Embora *bag-of-words* seja bem conhecido e utilizado ele sofre de muitos problemas, como:

- Para conjuntos de texto com muitas palavras distintas, a dimensão do vetor de saída para cada texto se torna demasiadamente grande. É comum acontecerem vetores de 100 mil posições por exemplo. Qualquer algoritmo de aprendizado tem dificuldades para encontrar padrões nesses casos;
- É perdida a informação que vem com a ordem das palavras, o contexto, pois o significado de uma frase pode mudar completamente de acordo com a ordem;
- Pontuação, palavras com caracteres em maiúsculo, palavras sem grande valor semântico, e variações da mesma palavra, como conjugações verbais, influenciam o vocabulário criado, entretanto há formas de lidar com esses casos.

Houveram pesquisas e tentativas de se propor um novo modelo de mapeamento de texto superando os problemas do *bag-of-words*, dentre elas as que mais fizeram sucesso e são utilizadas até hoje são as arquitetura propostas por Tomas Mikolov [13].

Estas arquiteturas utilizam como base uma de rede neural com uma camada escondida para obter uma representação vetorial de uma palavra, chamada *embedding*.

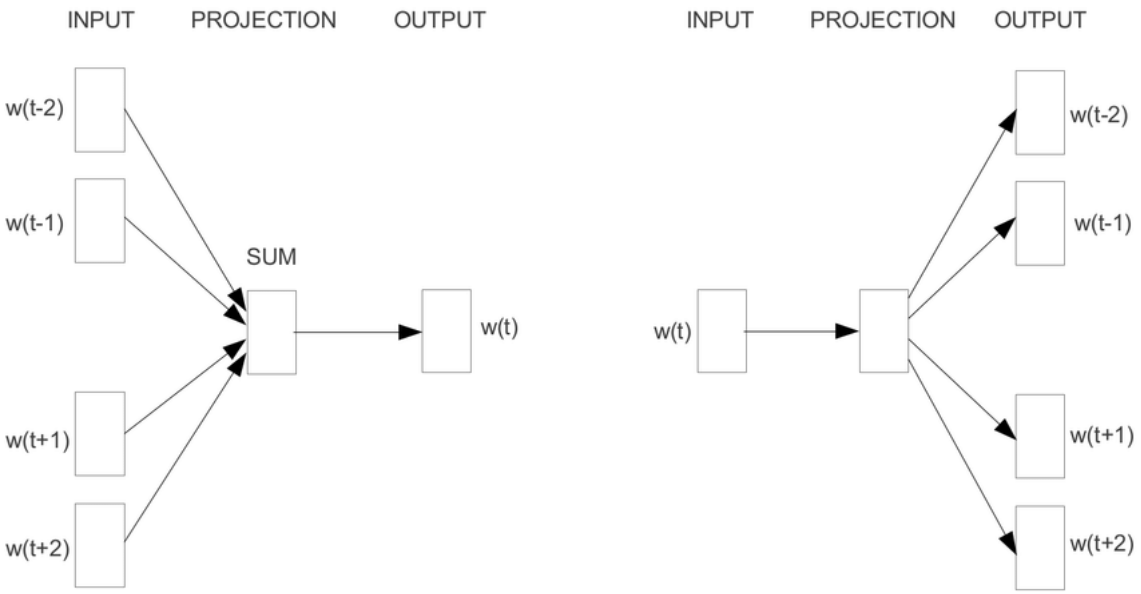


Figura 2.11 - Arquiteturas para mapeamento de palavras de Tomas Mikolov [13].

Os modelos de Mikolov se dividem em dois tipos [13]:

- Continuous Bag of Words - rede que tenta predizer palavras faltantes em uma sentença. Recebe como entradas quatro palavras, duas que vem antes e duas depois da palavra faltante;
- Skip-gram - rede que tenta predizer duas palavras que vem antes e duas palavras que vem depois de uma determinada palavra.

Com estas redes é possível obter uma representação vetorial de cada palavra de forma que cada dimensão do vetor resultante representa alguma forma de contexto que foi aprendido pela rede. Palavras com semântica parecida serão mapeadas em vetores próximos, algo que tende a ajudar os classificadores de texto, pois agora o classificador passa a ter percepção melhor do contexto. Geralmente são utilizados vetores de 50 a 1000 dimensões para cada palavra, o que também é um grande ganho de dimensionalidade em comparação ao *bag-of-words*.

As estruturas de Mikolov culminaram no uso de *embeddings* pelos métodos de processamento de linguagem atuais. Atualmente se utilizam outras estruturas, na maioria derivadas das apresentadas, como fasText e Glove [14, 15].

2.5 – Técnicas de balanceamento de corpus

Os modelos de aprendizado de máquina, tais como árvores de decisão, vizinhos mais próximos e redes neurais, sofrem com diversos problemas presentes nos dados. Um destes problemas é o mal balanceamento entre as diferentes de classes.

Pode acontecer que em um corpus existam duas classes A e B, sendo que B está presente apenas em 1% dos dados, logo se um modelo predizer a classe A para todas as instâncias ele obterá 99% de acurácia. Todavia será um modelo sem utilidade, já que não conseguiu de fato representar ganho em separar as classes.

Logo, há técnicas para amenizar os efeitos do mal balanceamento entre as classes. Essas técnicas se dividem em duas vertentes:

Oversampling: criar dados artificiais das classes minoritárias

Undersampling: selecionar dados das classes majoritárias

Entre os modelos de oversampling, um dos mais utilizados é o SMOTE, que consiste em criar pontos artificiais entre pontos reais numa classe minoritária. Para isto ele seleciona aleatoriamente um ponto no dataset, em seguida encontra os *N* vizinhos mais próximos deste ponto, então algum vizinho é selecionado aleatoriamente e o ponto final produzido estará em alguma posição aleatória entre o ponto de origem e o vizinho selecionado.

Dentre as técnicas de Undersampling podem ser destacadas [16]:

Randômica: Escolhe um conjunto de pontos aleatoriamente entre todos os pontos da classe majoritária;

Cluster: Utiliza KMeans para descobrir diversos centróides da classe majoritária e em seguida substitui os pontos reais pelos centróides;

NearMiss: A ideia desta técnica é selecionar o mínimo de pontos que permita diferenciar adequadamente as classes entre si. Para isto são usadas duas técnicas de forma simultânea: escolher pontos da classe majoritária que possuam a menor distância média

para pontos da fronteira da classe minoritária e que também possuam a menor distância para os pontos mais distantes da classe minoritária;

Tomek's links: A região em que se localiza a fronteira de decisão das classes é por vezes povoada de elementos de ambas as classes. O conceito links de Tomek foi criado por Ivan Tomek 1976. São basicamente pares de pontos de classes diferentes em que cada um é o vizinho mais próximo do outro. Esta técnica baseia-se em filtrar esses links, eliminando os pontos da classe majoritária;

EditedNearestNeighbours: Esta técnica atua filtrando instâncias da classe majoritária pela sua vizinhança. São descoberto os *N* vizinhos mais próximos de cada ponto da classe majoritária, e se a classe destes vizinhos não concordar com a classe do ponto selecionado, ele é então descartado;

InstanceHardnessThreshold: De certa forma é um método mais supervisionado, proposto em 2014. Neste método é necessário treinar um modelo de machine learning previamente, o modelo então prevê a probabilidade dos pontos da classe majoritária serem da classe de fato, os pontos que resultam em baixa probabilidade são descartados na filtragem.

As técnicas de Oversampling e Undersampling possuem vantagens e desvantagens quando comparadas entre si [17], como:

- Em Undersampling será perdida informação da classe dominante. Esta perda de informação pode gerar uma piora na performance do classificador;
- Os pontos produzidos por processos de Oversampling podem aumentar o efeito de overfitting, que ocorre quando o classificador não generaliza bem o comportamento dos dados de treinamento.

Alexander Liu pesquisou os efeitos das técnicas de Oversampling e Undersampling em *datasets* de textos desbalanceados em 2004 [17]. Ele testou alguns classificadores para diferentes conjuntos de textos desbalanceados. Ele demonstrou que a escolha da melhor técnica depende do funcionamento da técnica em si, do tipo de técnica (Oversampling ou Undersampling), porém também do classificador utilizado e também do conjunto de dados. Todavia em seus resultados é possível verificar que as técnicas de Oversampling apresentaram melhores pontuações que as de Undersampling na maior parte dos casos.

Para modelos tradicionais que representam os textos utilizando técnicas como *bag-of-words*, onde cada texto é convertido em um vetor de tamanho fixo, é simples utilizar as técnicas de balanceamento demonstradas acima, no entanto para redes neurais com *embeddings* o problema é mais complexo, pois cada texto será convertido em vetores de tamanhos distintos.

Existem algumas formas de lidar com o problema de mau balanceamento de classes com redes neurais. Algumas delas são:

- Adaptar a função de custo da rede para priorizar as classes que possuem menos instâncias;
- Utilizar uma rede GAN [18] para gerar novos dados das classes minoritárias.

2.6 – Redes neurais

2.6.1 – Introdução

Redes neurais, também chamadas de redes neuronais são estruturas matemáticas compostas por neurônios com objetivo de aproximar uma determinada função que mapeia os dados de entrada em uma determinada saída. Podem ser utilizadas para identificar o sentimento presente em um texto, o conteúdo em uma imagem, etc.

Os neurônios por sua vez são abstrações matemáticas que recebem entradas numéricas, aplicam uma função de ativação e geram uma determinada saída. O treinamento da rede, isto é, o ajuste de pesos entre as camadas é realizado através de uma função de custo, tal como a entropia cruzada [19], a técnica de atualização dos pesos mais comum é o *backpropagation*, que atualiza os pesos entre as diferentes conexões ao longo da rede usando o gradiente da função de custo.

Há diferentes tipos de neurônios e diferentes arquiteturas de redes. As redes MLP, Multi Layer Perceptron, são as mais antigas em termos de redes multicamadas, nelas podem ser encadeadas camadas com numerosos neurônios do tipo Perceptron. Embora esse tipo de rede possua boa capacidade preditiva, ele falha em alguns pontos, como:

- É difícil treinar uma rede do tipo com muitas camadas. Pois tende a ocorrer o problema de *vanishing gradient* ou o de *explosion gradient*, onde o

gradiente da função de custo pode assumir valores ínfimos ou muito altos para as últimas camadas da rede, impossibilitando o treinamento;

- Não lida tão bem com dados em sequências, principalmente sequências de tamanho variado;
- Não lida bem com dados oriundos dos pixels de imagens, a dimensionalidade das imagens é muito grande e a rede possui dificuldade em aprender padrões nessas condições.

Para tratar os problemas mencionados foram criadas diversas arquiteturas de redes para diferentes problemas, dentre elas podemos destacar a arquitetura BiLSTM, utilizada para aprendizado de sequências e CNN, utilizada para aprendizado de dados tabulares com alta dimensionalidade.

2.6.2 – Rede BiLSTM com atenção

As redes chamadas recorrentes - RNN - são um ramo recente da área de redes neurais. Elas surgem com objetivo de tratar de forma mais eficiente problemas de classificação de sequências. Essas sequências podem ser: evoluções temporais de sensores, linhas de texto, áudios, vídeos, etc.

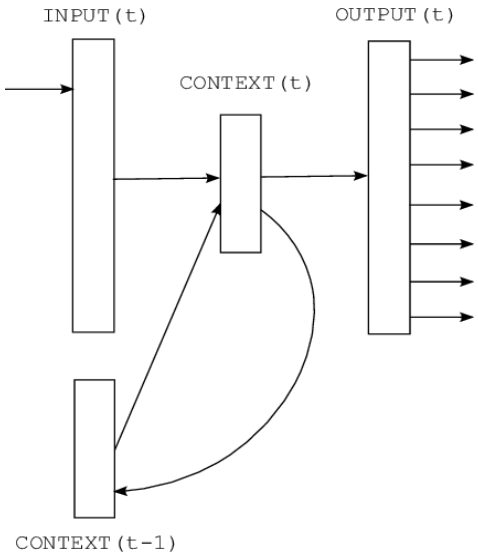


Figura 2.12 - Arquitetura geral de uma rede recorrente [20].

Na figura 2.12 é possível ver um diagrama geral das redes recorrentes, cada entrada ao longo de uma sequência leva a rede a um novo estado, todavia essa mudança de estado também depende do estado anterior, criando uma recorrência no processamento dos dados. O estado final da rede no fim da sequência é então repassado para camadas de saída.

Treinar um modelo recorrente esbarra em problemas com os métodos de otimização do gradiente, tornando uma tarefa árdua. O modelo RNN que conseguiu resolver estes problemas e se tornou a grande referência na área são as redes LSTM.

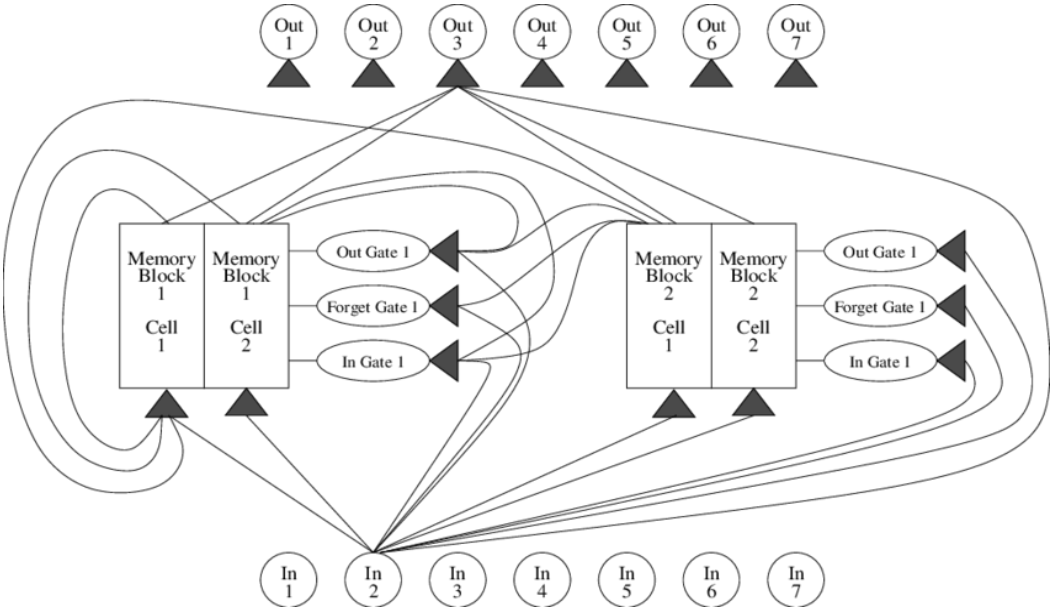


Figura 2.13 - Esquema geral de funcionamento de uma rede LSTM [21].

As redes LSTM são baseadas em células, como pode ser visto na figura 2.13. Cada entrada da série é sequencialmente repassada à célula da rede, esta por sua vez possui *gates*, que são estruturas responsáveis por regular a atualização do estado de saída, os *gates* devem tratar: quando o estado deve ou não ser alterado, quais partes são alteradas e com qual intensidade.

Este tipo de rede obtém bom desempenho no aprendizado de sequências, e também não é tão afetada pelos problemas de atualização do gradiente, devido à estrutura interna de suas células. Todavia ainda há espaço para melhoria, esta rede tem problemas em alguns pontos, dentre eles:

- A saída da rede sempre está no último estado, logo se a sequência é muito longa e a informação relevante está no início por exemplo a rede vai ter de propagar esta informação até o último estado;
- Não fica claro qual seria a melhor direção de leitura dos dados da sequência analisada, em determinados casos pode ser também relevante ler a informação na ordem inversa.

Para tratar estes problemas surgiram variações do LSTM, a rede que se destacou nesse ponto ficou conhecida como BiLSTM com atenção.

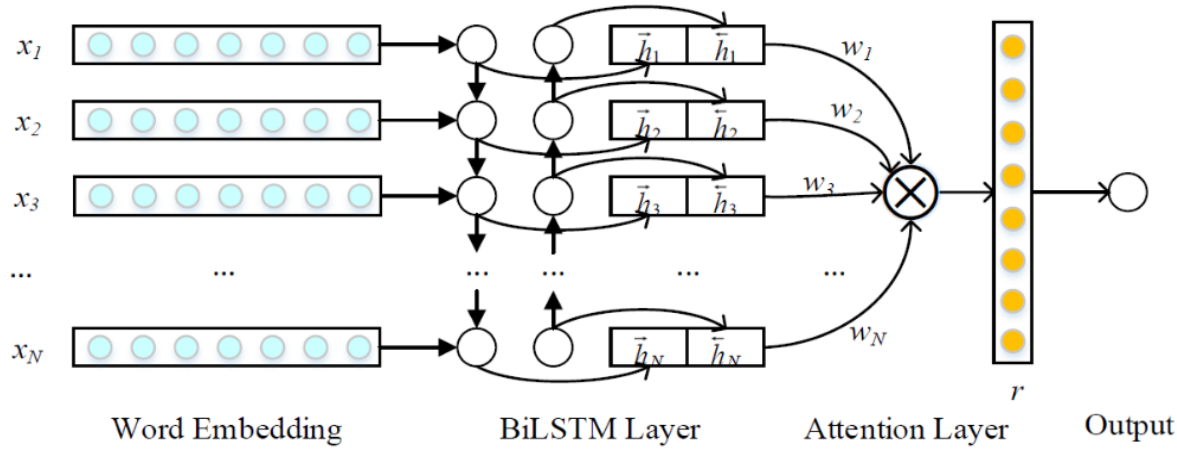


Figura 2.14 - Esquema de funcionamento de uma rede BiLSTM com atenção [22].

Como se pode observar na figura 2.14, a rede BiLSTM atua utilizando exatamente duas redes LSTM que lêem as informações da entrada em sentidos opostos. O estado resultante da rede BiLSTM é considerado como sendo a concatenação dos estados de ambas redes internas. O mecanismo de atenção é utilizado para que a rede possa produzir uma análise utilizando todos os estados intermediários e não apenas o final, pois esta camada recebe todos estados e pondera o resultado final dando pesos diferentes a cada estado intermediário.

2.6.3 – Rede CNN

As redes convolucionais - CNN - surgem com o objetivo de tornar a classificação de dados tabulares de alta dimensionalidade mais assertiva. Esse tipo de rede necessita aprender padrões de entradas de altas dimensões, como imagens. Elas têm obtido grande destaque e evolução nos últimos anos, principalmente graças ao ImageNet Challenge [23], competição anual no tema de classificação de imagens.

Uma rede MLP tem dificuldades em processar dados como imagens, dado que cada pixel da imagem representará uma variável de entrada, a dimensionalidade tende a ser alta inviabilizando o aprendizado da rede. Uma forma de contornar isto é criando extratores de features, ou seja, mecanismo que gerem features preditivas a partir dos dados de entrada, porém tal processo não é simples.

Os modelos convolucionais possuem diversas camadas onde são realizadas convoluções e reduções de dimensionalidade. As convoluções atuam como extratores de features automáticos, ou seja, elas mapeiam os pixels de entrada em mapas de características (*features*). Estes extratores também são treinados junto com a rede em si, logo os padrões aprendidos são automáticos e só dependem do problema e dos dados de entrada.

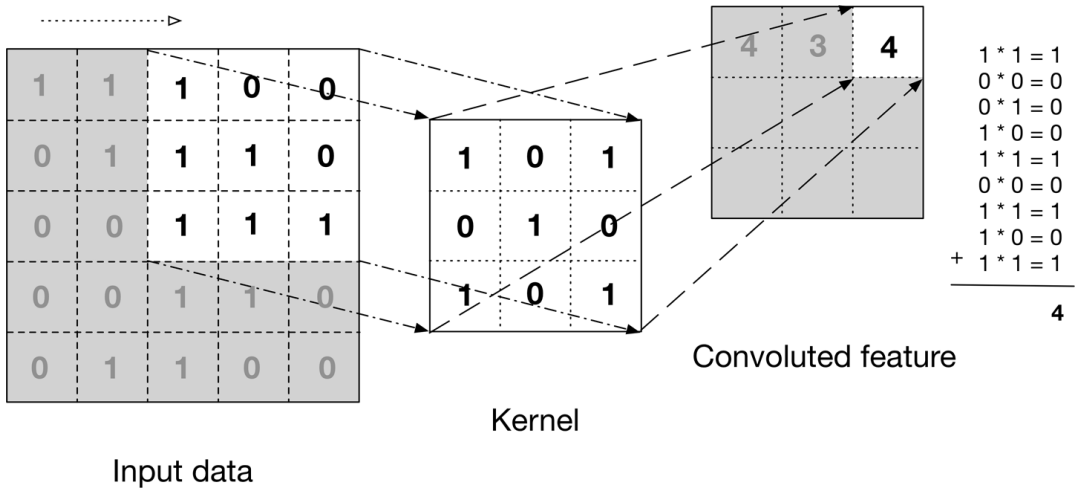


Figura 2.15 - Exemplo de uso de kernels em redes CNN [24].

Na figura 2.15 pode-se visualizar a estrutura de um extrator de características, também chamado de *kernel*, que é usado nas camadas convolucionais das redes CNN. Os *kernels* são matrizes responsáveis por convoluir um quadro em características, de modo que cada um atua como uma janela que desliza sobre o quadro de origem calculando

convoluções a cada passo. O tamanho do *kernel* ditará os tipos de características extraídas, *kernels* maiores extraem características de maior volume, ou seja, que compreendem maiores regiões do frame de entrada.

A camada de convolução é sempre sucedida por uma camada que aplica funções de ativação nos dados das matrizes geradas. Neste contexto a função comumente utilizada é a ReLU, denotada a seguir.

$$r(x) = max(0, x)$$

O próximo passo é reduzir a dimensionalidade das matrizes geradas, para isto é utilizada a técnica de *max-pooling*, ilustrada na figura 2.16, nesta técnica cada matriz de entrada é subdivida em matrizes menores, em seguida é obtido o valor máximo entre os valores de cada submatriz, a partir disto é montada uma nova matriz apenas com os resultados. A ideia é preservar apenas as informações mais relevantes.

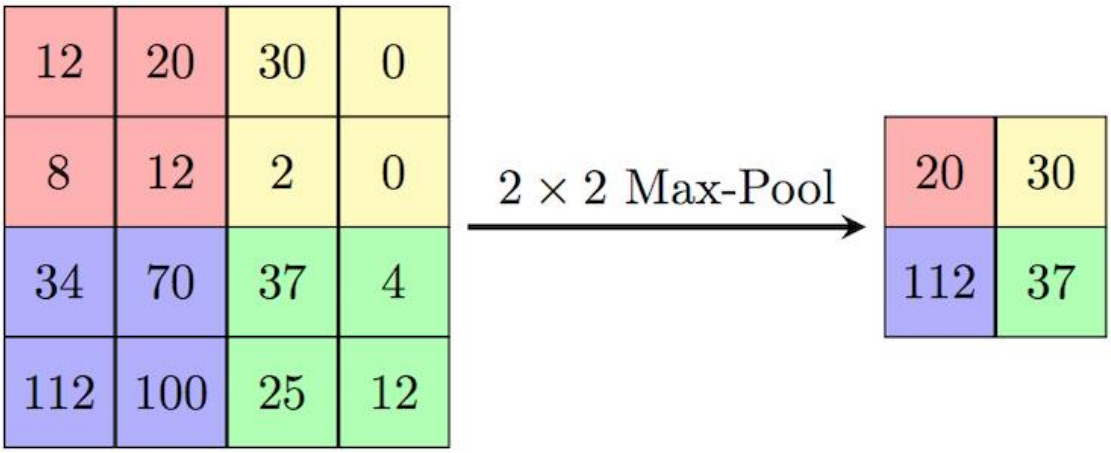


Figura 2.16 - Exemplo de uso de *max-pooling* em redes CNN [25].

Diversas camadas convolucionais e de *pooling* são empilhadas nestas redes. De forma que a extrair características dos dados e resumir a informação de saída a cada passo. No fim, os mapas finais são transformados em único vetor que servirá como entrada para uma camada densa, geralmente uma MLP.

Dentre as diversas aplicações destas redes, elas também podem ser utilizadas para classificação de sentenças, considerando a composição de todos os *embeddings* de uma determinada sentença como um quadro de entrada para a rede. Tal processo está demonstrado na figura 2.17.

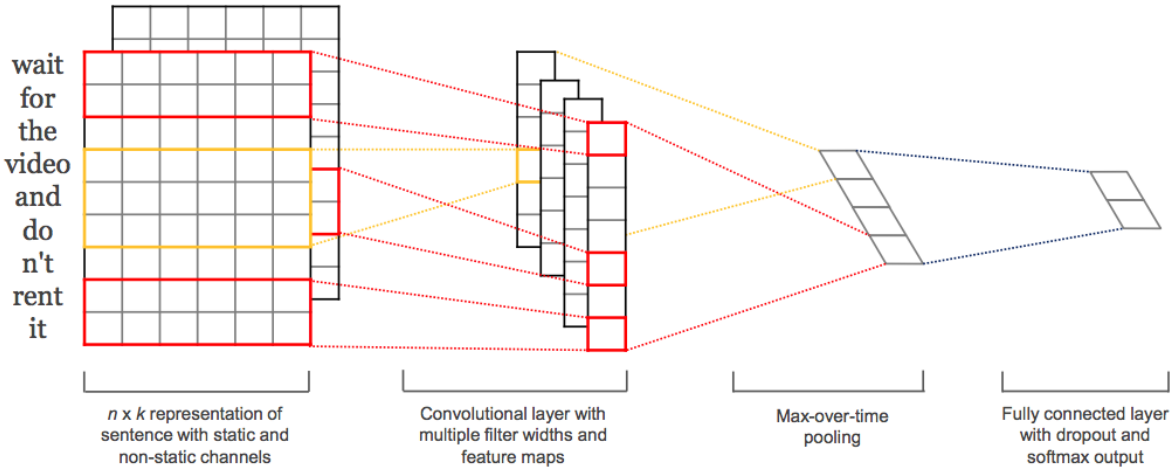


Figura 2.17 - Esquema de funcionamento de uma rede CNN para classificação de sentenças [26].

2.7 – Métodos de avaliação

Para estimar o comportamento de um modelo preditivo é comum utilizar técnicas de validação dos resultados, nas quais o modelo é treinado em um conjunto de dados, chamado conjunto de treino e avaliado em outro conjunto, chamado de conjunto de teste. Podem acontecer casos em que o modelo se super ajusta aos dados, situação conhecida como *overfitting*, na qual o desempenho é pior no conjunto de teste do que no de treino.

Para modelos simples a validação cruzada de dez ciclos é comumente empregada, neste método todo o conjunto de dados é dividido em dez subconjuntos e então são realizadas dez interações. Em cada interação o conjunto de teste será representado por um subconjunto de forma alternada e o conjunto de treino será composto pelos restantes. Com isto, o modelo será avaliado sobre dez conjuntos de teste e as medidas de interesse, como a acurácia, poderão ser obtidas através da média ao longo das interações.

Para modelos complexos, como redes neurais, já é comum utilizar apenas uma validação mais simples, devido ao alto tempo necessário para treinamento. Uma forma de fazer esta validação é dividir todo o conjunto de dados em um conjunto de treino e um de teste, geralmente em proporções 80/20 ou 70/30 e retirar as medidas de interesse.

Dependendo do tipo de modelo diferentes medidas podem ser tomadas para avaliação, caso o *target* seja numérico por exemplo pode-se utilizar a raiz do erro médio quadrático, entretanto para um classificador as principais medidas são: acurácia, precisão, cobertura e a medida F1. A acurácia representa a assertividade geral do modelo, as outras medidas representam o comportamento do modelo para cada classe.

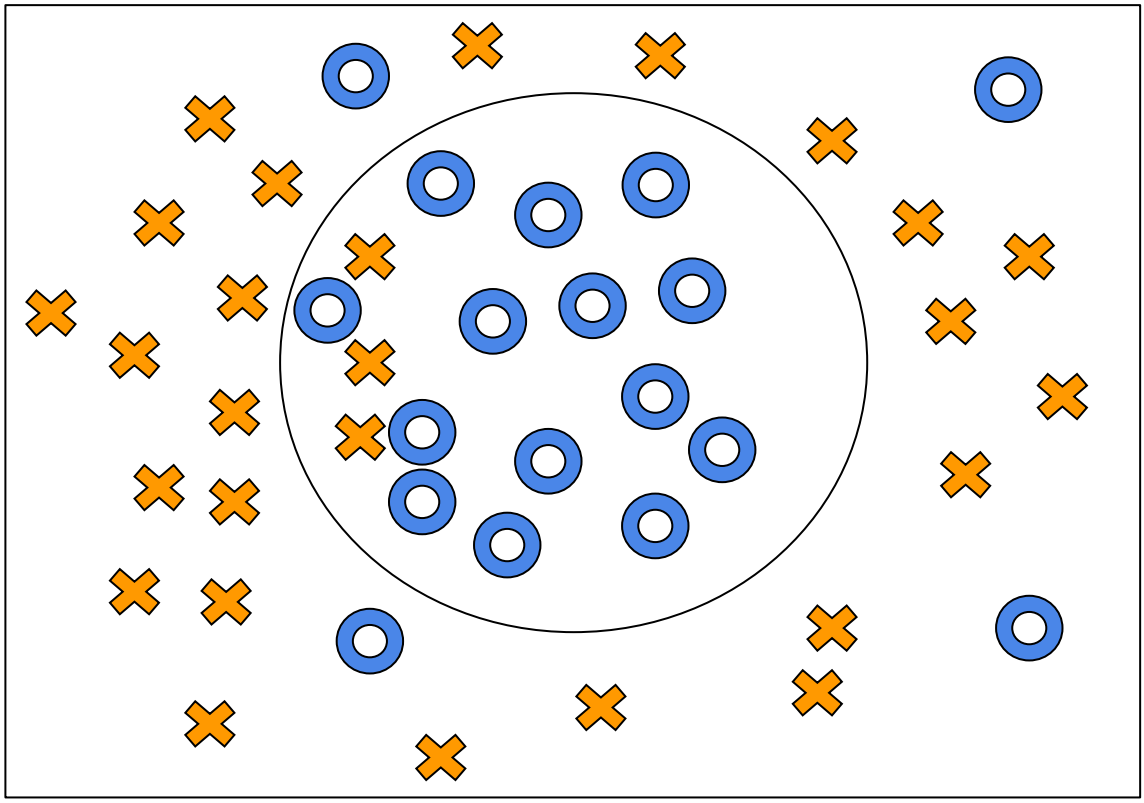


Figura 2.18 - Exemplo de fronteira de decisão de um classificador.

Na figura 2.18 podemos ver a fronteira de decisão de um dado classificador, há duas classes: “O” e “X”, o modelo preditivo treinou com um dado conjunto e definiu que o elipse no centro é a área onde estão os pontos da classe “O” e o restante do espaço é representado pela classe “X”. A partir da figura pode-se perceber que para esta fronteira haverão pontos:

- Da classe “O” no espaço interior da elipse de decisão, chamados de verdadeiros positivos (VPs);
- Da classe “X” no espaço interior da elipse de decisão, chamados de falsos positivos (FPs);
- Da classe “X” no espaço exterior da elipse de decisão, chamados de verdadeiros negativos (VNs).
- Da classe “O” fora da elipse de decisão, chamados de falsos negativos (FNs).

Com isto em mente podemos definir matematicamente as métricas de interesse, como pode ser visto na figura 2.19. Portanto é perceptível que acurácia de fato representa a assertividade geral, enquanto que a precisão é uma medida por classe e representa a porcentagem de pontos classificados corretamente dentre todos os pontos mapeados na

classe em questão, a cobertura por outro lado representa a porcentagem de pontos de uma determinada classe que foi corretamente identificada, e por fim a medida F1 de uma classe é a média harmônica entre sua precisão e cobertura.

$$\text{Acurácia} = \frac{VP + VN}{VP+FP+VN+FN} \qquad \text{Precisão} = \frac{VP}{VP + FP}$$

$$\text{Cobertura} = \frac{VP}{VP + FN} \qquad \text{Medida F1} = \frac{2 * \text{precisão} * \text{cobertura}}{\text{cobertura} + \text{precisão}}$$

Figura 2.19 - Definição equacional de métricas de interesse na avaliação de um classificador.

O critério usado para priorizar as medidas relevantes depende do problema e dos dados em questão. A precisão e a cobertura de classes minoritárias por exemplo tendem a ser menor que as das maioritárias, sendo necessário o uso de técnicas de balanceamento para minimizar o problema.

Capítulo 3

Elaboração do dataset para classificação

3.1 – Seleção do conjunto de dados

Toda e qualquer técnica de aprendizado de máquina supervisionado demanda um conjunto de dados para ser usado no treinamento. Ou seja, dados que permitam ao modelo criar estruturas para realizar predições em cima de novos dados.

Para o presente trabalho foram observados diversos conjuntos de dados em Português Brasileiro, tais como:

- Consumer Business Complaints in Brazil [27]
  Dados sobre reclamações de clientes brasileiros ao Procon nos anos de 2012 a 2016. Há detalhamento sobre todas as solicitações no período, incluindo informações sobre: descrição do problema, descrição do assunto, se foi resolvido, dados da empresa associada e da localidade;
- Wikipedia page in brazilian portuguese [28]
  Conjunto de páginas da Wikipédia na linguagem local. Há uma grande quantidade de artigos, porém não há uma boa estrutura, só está disponível o texto principal de cada artigo;
- Brazilian E-Commerce Public Dataset by Olist [29]
  Dados sobre E-Commerce brasileiro nos anos de 2016 a 2018. Inclui informação sobre cerca de 100 mil vendas, contando com comentários dos consumidores, dados de anúncios, detalhes da venda em si, dos itens comprados, etc;
- Fake news Br [30]
  Seleção de notícias falsas e reais retirada de jornais online brasileiros. Coleção criada para um projeto da USP sobre identificação de notícias falsas;

- Tweets_eleicao2018 [31]

  Cerca de 78 mil tweets sobre a eleição presidencial brasileira de 2018. São textos curtos que contam com *hashtags* que os associam às eleições da época;
- Song lyrics from 6 musical genres [28]

  Conjunto de 164 mil letras de música retiradas do website Vagalume de seis diferentes gêneros em diferentes linguagens. Criado por um estudante de estatística com objetivo de usar redes neurais adversárias para produção de novas letras musicais;
- MilkQA Dataset [33]

  Coleção de aproximadamente 2600 perguntas e respostas reunidas pela Embrapa sobre agricultura e afins. As perguntas são de autores distintos com diferentes níveis de conhecimento sobre os assuntos, as respostas são de especialistas da Embrapa sobre os assuntos requisitados;
- LeNER-Br: a Dataset for Named Entity Recognition in Brazilian Legal Text [34]

  Conglomerado de 50 textos jurídicos brasileiros mapeados em diferentes entidades. Ou seja, cada palavra de cada texto foi mapeado em uma entidade dentre as a seguir: Pessoa, Localização, Tempo, Organização e Nada.

Entre os *datasets* apresentados, aqueles que se adequam diretamente à tarefa de classificação de documentos são: “Fake news Br” e “Song lyrics from 6 musical genres”, já que o problema de classificação de documentos poderia ser ramificado nos problemas de seleção de notícias falsas e seleção de gênero musical a partir da letra.

Detectar notícias falsas é uma necessidade para a sociedade brasileira, dado ao alto volume de notícias do tipo que são difundidas nas redes sociais. Porém, a quantidade de notícias reunidas pelo projeto “Fake news Br”, originário da USP, é pequena e dificulta a confecção de um modelo preditivo. A grande dificuldade de reunir notícias confirmadamente falsas também impossibilitou atacar este problema. Há diversas plataformas brasileiras que classificam notícias falsas, entretanto nenhuma delas dispõe da notícia falsa na íntegra em seus artigos de forma estruturada. Mesmo após contato, nenhuma delas se prontificou a disponibilizar os dados das notícias.

O problema de identificar o gênero musical a partir da letra é facilmente mapeado para um problema de classificação textual, onde busca-se identificar uma propriedade subjacente de um texto através de um histórico de casos semelhantes. E cumpre o requisito de possuir uma grande quantidade de dados para aprendizado.

Os dados extraídos estavam hospedados no Kaggle [32], onde é possível visualizar de forma preliminar que para apenas 6 gêneros existem 164.790 letras musicais disponíveis.

Desse modo, o *dataset* escolhido foi o de gêneros musicais. Este foi reunido por Anderson Neisse, aluno de mestrado da Universidade Federal de Viçosa [35]. Neste conjunto de dados há letras de várias línguas distintas, logo foi optado pela língua Português Brasileiro. Portanto, o subproblema a ser atacado é classificação de gênero músical a partir da letra de uma dada música em português.

3.2 – Construção do dataset

O *dataset* de gêneros musicais foi construído a partir do Vagalume [1]. Para tal o mestrando Anderson desenvolveu um *scrapper* [35] na linguagem R. Este *scrapper* faz *download* de todas as letras de músicas do Vagalume de seis diferentes gêneros previamente selecionados. Também são extraídos outros dados como popularidade do artista, nome do artista, gêneros do artista, etc.

Embora houvesse uma boa quantidade de registros já disponíveis, optou-se por desenvolver e aperfeiçoar o *scrapper* de forma que ele pudesse ser utilizado para baixar toda a base de letras de todos os gêneros disponíveis, cerca de 80 gêneros e 390.000 letras musicais distintas. E ainda, identificar a linguagem daquela letra com base no mapeamento prévio feito pelo próprio Vagalume.

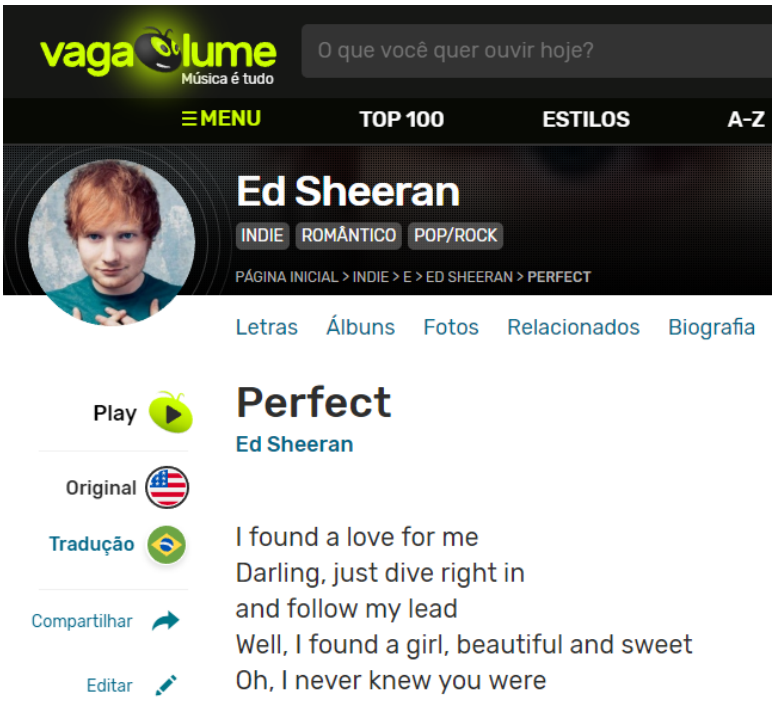


Figura 3.1 – Perfil do artista Ed Sheeran no Vagalume [1].

No Vagalume cada artista está listado na página de um estilo musical, entretanto no perfil do artista também aparecem outros estilos, podendo chegar a três estilos por artista. Na página do perfil também é possível visualizar uma lista com links para todas as letras, e ao clicar em uma delas é aberta a página correspondente com a letra, e em alguns casos há uma marcação do Vagalume com uma bandeira, que indica o idioma da música, como pode ser visto na figura 3.1.

O *scrapper* original já trazia os campos: letra da música, nome do artista, popularidade do artista, lista de gêneros do artista, entre outros, para seis gêneros musicais. A nova versão do *scrapper*, desenvolvida para esta monografia, contou com modificações e correções, dentre elas:

- Passaram a ser utilizados todos os gêneros, cerca de 80, em vez de apenas 6;
- Foi necessário identificar e filtrar letras musicais duplicadas;
- Tratamento para caracteres fora do padrão UTF-8;
- Obtenção do idioma identificado pelo Vagalume para cada letra;
- Detecção do idioma da letra musical através de estratégia específica;
- Correção para o caso em que não há informação de popularidade do artista;
- Tratamento para o caso em que mesmo havendo o link, não existe a página da letra da música;
- Compilação e salvamento do resultado final em CSV.

Um fluxograma de execução para o *scrapper* pode ser visualizado na figura 3.2.

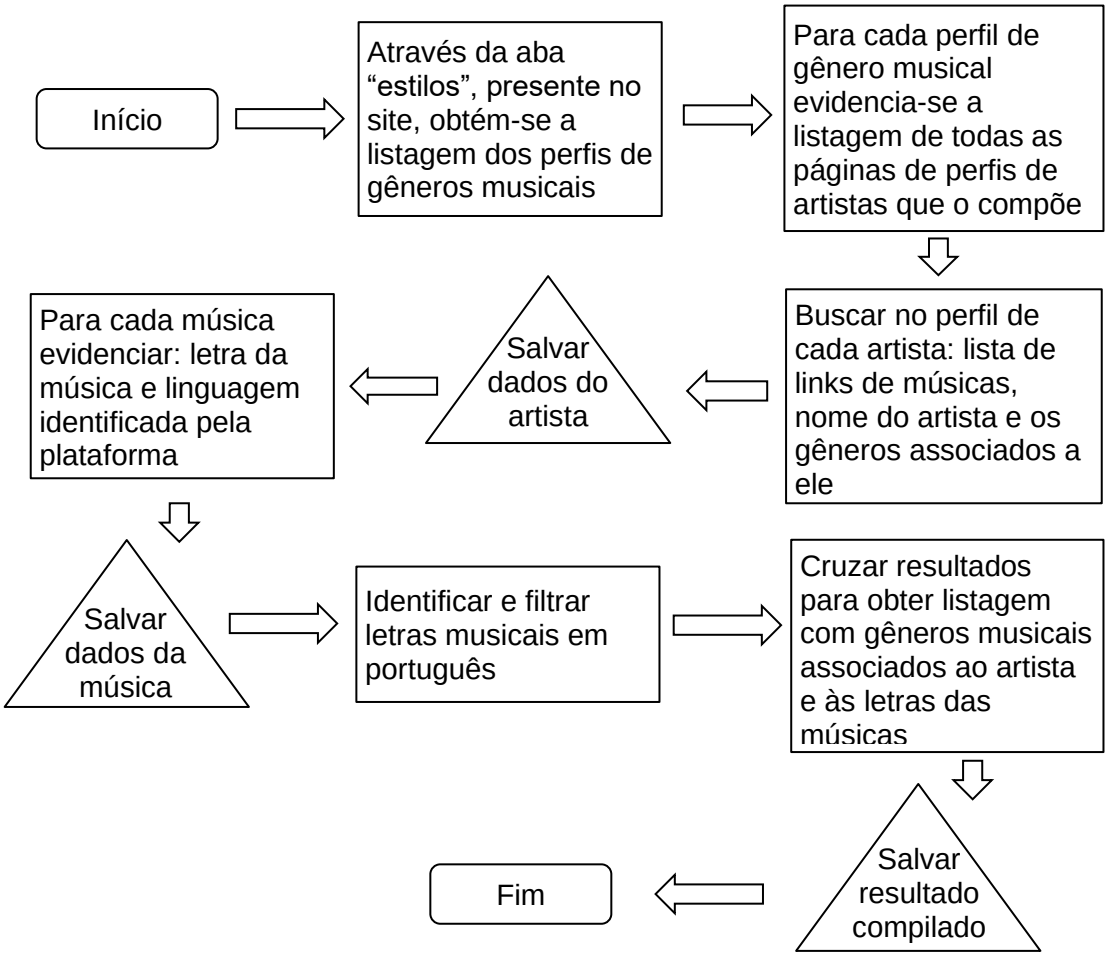


Figura 3.2 – Fluxograma da execução do scrapper.

A linguagem identificada pelo Vagalume para cada letra foi obtida através da bandeira disponível na página da letra, assim como na figura 3.1. É possível obter através do HTML uma classe CSS que representa a bandeira. O Vagalume mapeou a linguagem de boa quantidade de letras musicais, porém não de todas, uma vez que nem sempre a bandeira está disponível. Logo, esta informação serviu de filtro e caso a bandeira existisse e fosse de outro país que não fosse o Brasil, a linguagem era considerada não adequada e a música retirada do *dataset* final. Fica claro que esse processo carece de assertividade. Na figura 3.3 pode-se perceber a proporção entre os idiomas previamente identificados pelo Vagalume nas letras baixadas.

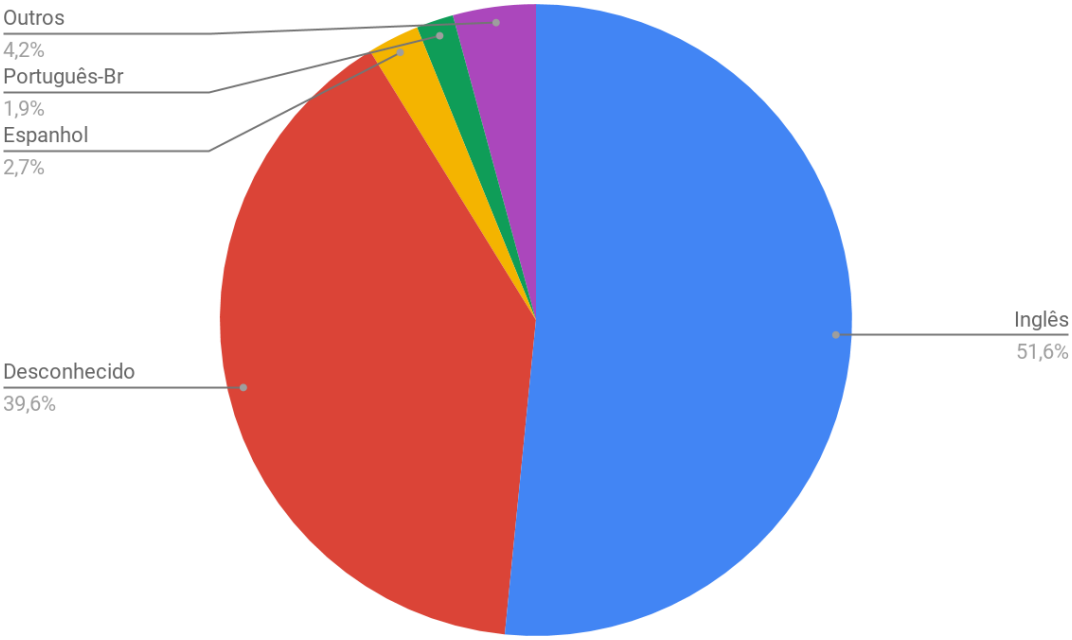


Figura 3.3 – Proporção de idioma identificado pelo Vagalume entre as letras musicais.

Para detectar o idioma efetivo da letra musical foram realizadas tentativas com algumas bibliotecas. A primeira foi com a biblioteca langdetect , que é baseada na biblioteca Java language-detection, esta por sua vez, foi elaborada pelo Google e atingiu alto desempenho na detecção de múltiplos idiomas [36]. Embora parecesse promissora, a biblioteca apresentou resultados instáveis, pois o número de letras identificadas como do idioma português mudava drasticamente para diferentes execuções. Após pesquisa de outras ferramentas da área, foi optado por testar a biblioteca TextBlob [37], outra ferramenta de referência. Entretanto também foi considerada inadequada, pois fazia chamadas para uma API de detecção de linguagens do Google [38], e esta por sua vez só permitia o uso gratuito de um número limitado de tentativas diárias. Por fim, o módulo que melhor se comportou foi o Polyglot [39], com ele foi adotada a estratégia: deixar apenas no *dataset* letras musicais onde só foi detectada a linguagem portuguesa e nenhuma outra. Tal preocupação é necessária pois é comum haver letras musicais com vários idiomas misturados.

3.3 – Análise exploratória

3.3.1 – Descrição do dataset

O dataset gerado possui diversos campos, todos estão listados na tabela a seguir:

Tabela 3.1 – Detalhamento de campos do dataset gerado pelo scrapper.

Nome do campo	**Descrição**
Artist	Nome do artista
Popularity	Popularidade do artista
Alink	Link do perfil do artista
ParentGenre	Gênero em que o artista está listado
Genres	Gêneros que aparecem na página do artista
SName	Nome da música
SLink	Link da página da música
Lyric	Letra da música
LanguageFlag	Idioma mapeado pelo Vagalume

Os campos de interesse para a classificação de documentos são: Letra da música (Lyric), gênero que o artista está listado (ParentGenre) e gêneros que aparecem na página do artista (Genres). A seguir estão alguns exemplos de instâncias do dataset apenas com os campos de interesse.

Tabela 3.2 – Exemplos de letras musicais presentes no dataset.

Lyric	**ParentGenre**	**Genres**
Minha princesa eu troco tudo pra estar conti...	Black Music	Black Music, Hip Hop, Rap
A história se iniciou em 1977, com uma bela mo...	Hip Hop	Hip Hop, Rap, Gospel/Religioso
Tudo bem meu bem mas eu tô indo. Dar aquele ro...	Soul Music	Soul Music, Funk, Black Music

Eu miro ao léu. Um traço no céu. Seu jato. Des...	World Music	World Music, Black Music, Pop
Simbora, vamo que hoje a noite me promete. Pra...	Pop	Pop, Funk Carioca, Black Music
Tanto tempo já vai caminhando. E ainda me pego...	Dance	Dance, Pop, R&B
Se ficar assim me olhando, me querendo, procu...	R&B	R&B, Black Music, Soul Music
Se eu disser que eu não quero crescer. Eu tô...	Blues	Blues, Black Music, Rap
Chega de bobeira. Chega de tanta besteira. Che...	Pop/Rock	Pop/Rock, Black Music, Gospel/Religioso
Corre na frente que atrás. Ficou gente sentada...	Samba	Samba, Hip Hop, Black Music

O conjunto de letras musicais original possuía exatamente 390.400 itens. Após a filtragem por idioma descrita na seção 3.3.2 sobraram 157.912 itens, cerca de 40% do total.

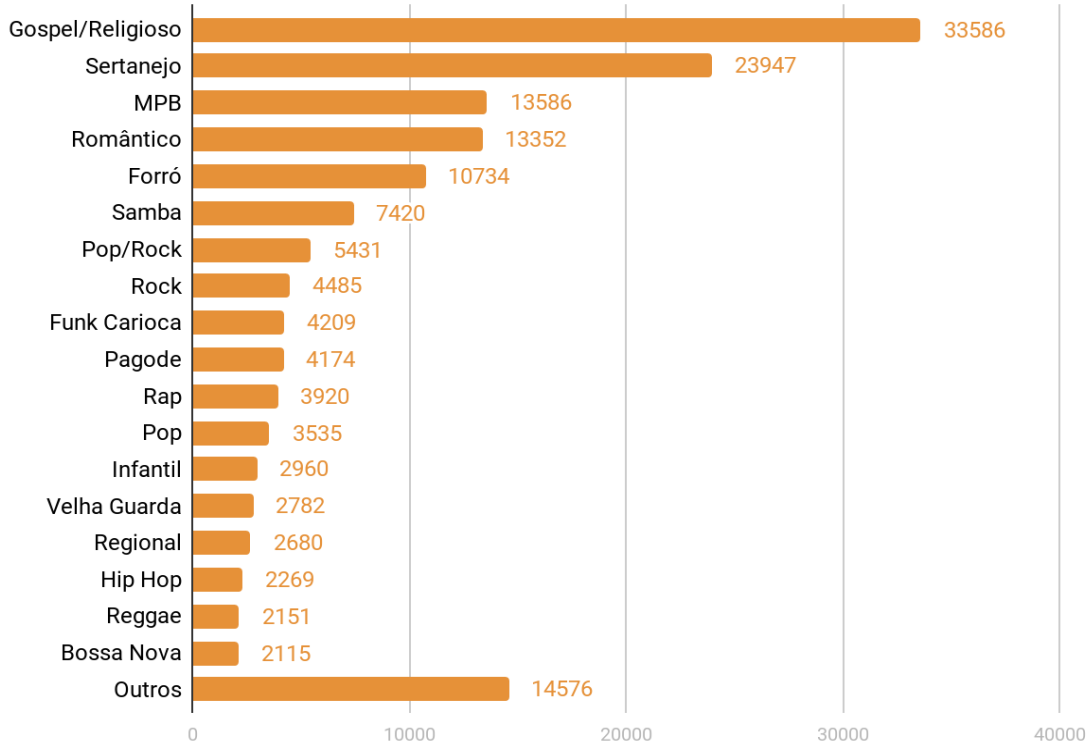


Figura 3.4 – Distribuição da variável ParentGenre entre as letras musicais do dataset.

Na figura 3.4 é possível visualizar uma distribuição da variável ParentGenre. Que em suposição representa o gênero principal do artista e também da música, por consequência.

Já na figura 3.5 é possível visualizar um histograma da quantidade de palavras da música ao longo do dataset. A distribuição é aproximadamente gaussiana, logo não é tão importante se preocupar com a questão de treinar/predizer com textos de tamanhos muito distintos. Cerca de 96% das músicas possuem menos de 400 palavras e a mediana da distribuição é de 131 palavras.

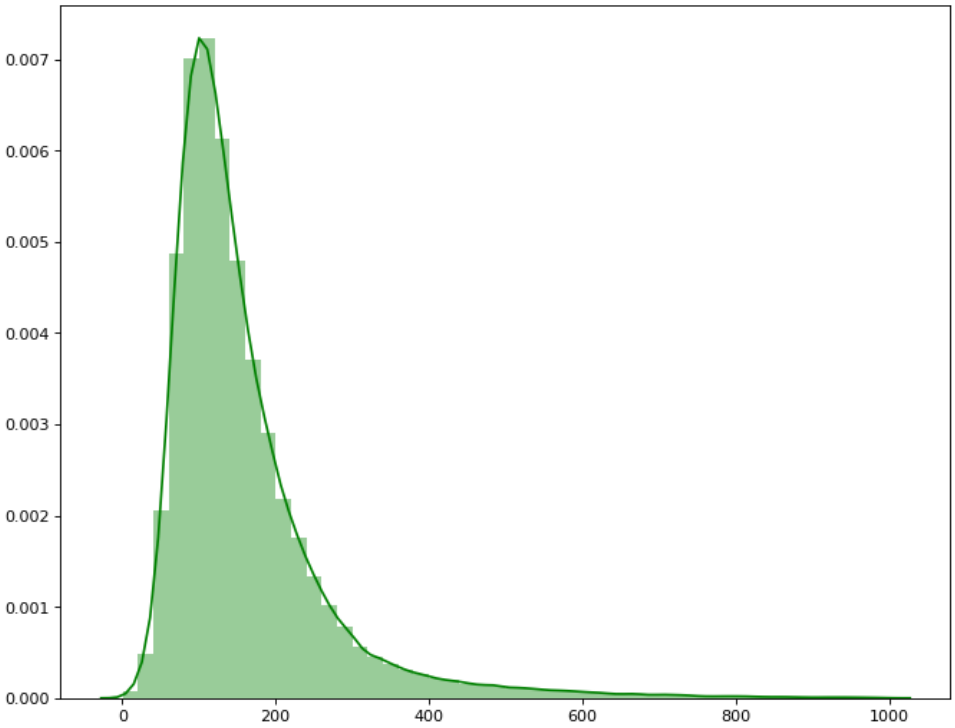


Figura 3.5 – Distribuição do número de palavras entre as letras musicais do dataset.

3.3.2 – Estudo do gênero musical

É importante reparar que a informação do gênero não é dada por música, o Vagalume apenas possui o mapeamento por artista, isto ocorre também para os sites concorrentes do Vagalume, não foi encontrado nenhum portal nacional de letras musicais que possua o gênero por música.

Para a classificação posterior será necessário definir um gênero por letra musical, e pelas limitações dos dados este gênero será algum dos listados pelo Vagalume. Portanto, ocorrerá o problema de existirem letras musicais que serão atreladas a um gênero

incorreto, para o caso em que um mesmo artista atua em diferentes estilos. Como por exemplo o caso do artista Marku Ribas, ele está listado no gênero Samba, entretanto também atua nos gêneros Rock e Funk, logo será escolhido apenas um gênero para todas as suas músicas, isto fará com que algumas letras musicais dele sejam incorretamente mapeadas.

Para lidar com esse problema, será levado em conta a hipótese de que estes casos devem ocorrer com estilos que, na maioria das vezes, não devem ser muito distintos uns dos outros, como por exemplo: Rock e Rock Alternativo. Parece plausível esperar que um artista se mantenha em torno de alguns estilos parecidos, de forma que mesmo que algumas músicas sejam mapeadas erroneamente isto não afete tanto o processo de classificação.

Outro grande problema é que há múltiplas classes por música. Sendo assim, é necessário definir um critério para escolher o gênero adequado para cada música, de forma a possibilitar uma classificação posterior. Foram analisadas algumas estratégias para fazer este mapeamento:

1. Filtrar o dataset, escolhendo apenas as músicas que tem apenas um gênero;
2. Considerar toda permutação de gêneros como uma classe distinta;
3. Tomar a coluna ParentGenre como o gênero principal e utilizar estratégias para lidar com os possíveis problemas.

A primeira estratégia funciona e é simples, porém sobram apenas cerca de 10.000 letras musicais para o uso, grande parte do dataset é perdido, assim como é possível visualizar na figura 3.6.

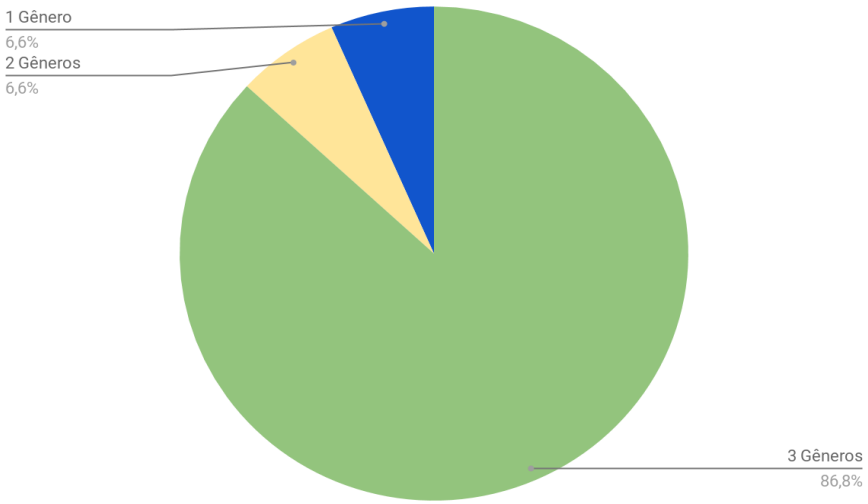


Figura 3.6 – Distribuição do número de gêneros entre as músicas do dataset.

A segunda estratégia cria uma grande quantidade de classes, cerca de 1867 possíveis permutações. Nessas permutações nascem classes com pouquíssimas instâncias.

Já a terceira estratégia toma a coluna ParentGenre como gênero principal do artista, este gênero é onde o artista está listado de fato e também sempre é o primeiro gênero da coluna Genres. Deste modo, parece que dentre os múltiplos gêneros mapeados pelo Vagalume há uma hierarquia, sendo o primeiro o mais relevante.

Seguindo pela terceira estratégia nos deparamos com outros problemas. Assim, como pode ser visualizado na figura 3.4, a maioria dos gêneros possuem poucas músicas em português. Portanto optou-se por considerar apenas os gêneros com no mínimo 3000 letras musicais associadas.

Fora este problema também foi realizado um estudo sobre os demais gêneros atribuídos a cada música, pois poderiam existir gêneros que sempre aparecem em conjunto e isto atrapalharia o classificador. Para tornar visível o problema, foi criada uma matriz de frequências relativas para mapear as frequências de cada par de gêneros. Uma parte da matriz está representada na figura a seguir como um mapa de calor. Não foi trazida toda a matriz pois é demasiadamente extensa, só foram deixadas as linhas e colunas de interesse.

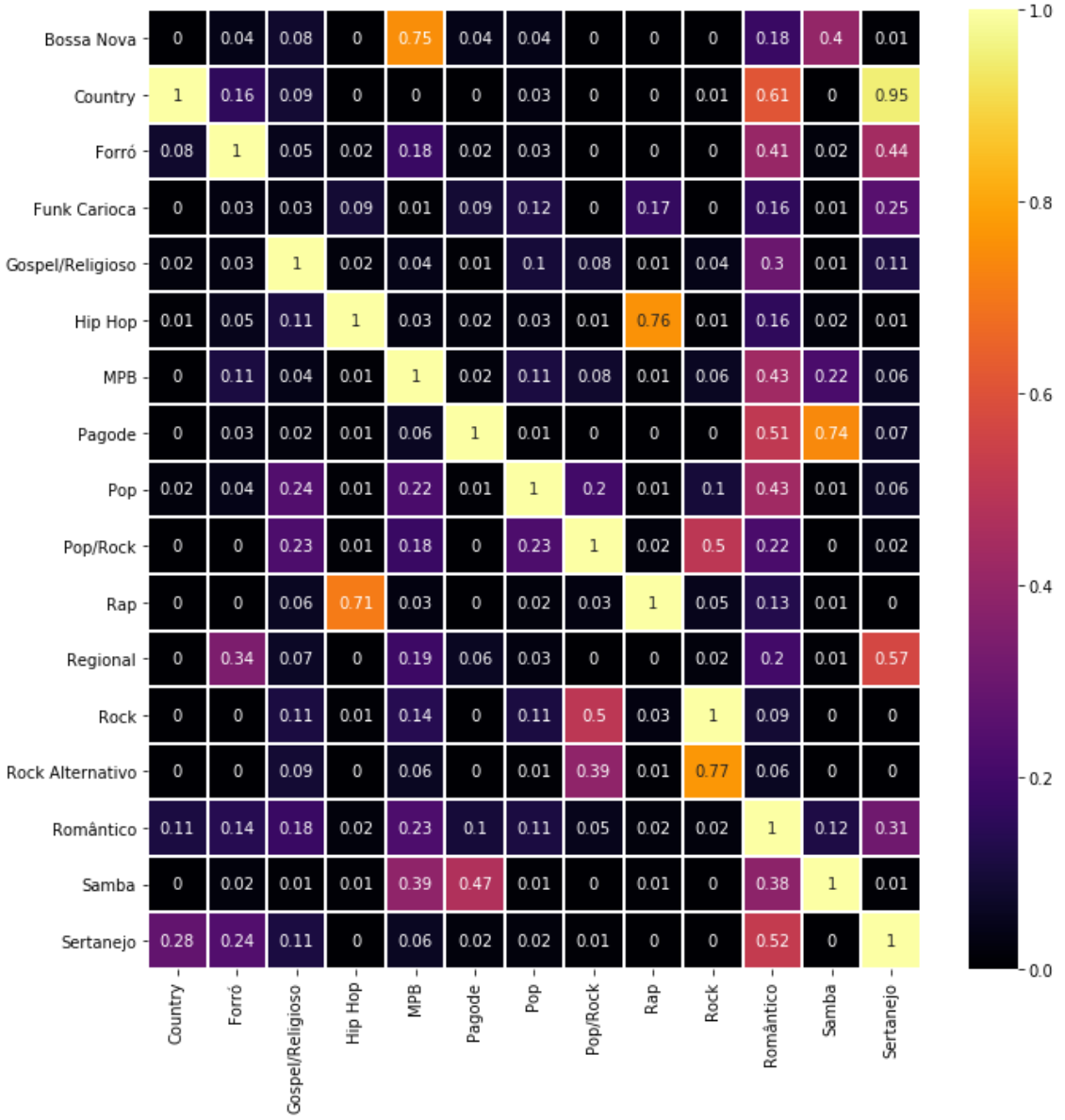


Figura 3.7 – Mapa de calor das frequências relativas dos pares de gêneros de interesse do dataset.

Cada linha representa o gênero pai e cada coluna o gênero filho. Por exemplo: olhando a linha de Pagode com a coluna Samba encontramos 0.74, este valor indica que 74% das letras de Pagode possuem Samba como gênero secundário ou terciário, por outro lado olhando a linha Samba e coluna Pagode encontramos 0.47, ou seja 47% das letras de Samba possuem Pagode como gênero secundário ou terciário.

A partir dessa matriz alguns pares ou conjuntos de gêneros foram selecionados para serem unidos em um só, a partir do critério: pares de gêneros que possuam frequência relativa maior que 0.4 em qualquer sentido serão considerados indistiguíveis, e portanto representados por uma classe só. Com isto, os seguintes grupos foram criados:

1. Samba e Pagode.
2. Rock, Pop/Rock e Rock Alternativo
3. Rap e Hip-Hop
4. Bossa Nova e MPB
5. Regional, Sertanejo, Forró e Country

É perceptível que o gênero Romântico possui comportamento diferente dos demais, pois na maior parte dos outros gêneros é bem comum aparecer Romântico como gênero secundário ou terciário. Assim sendo, levantou-se a hipótese de que talvez Romântico não seria um gênero propriamente dito e sim uma característica dos gêneros em si. Por simplicidade, foi optado por remover da seleção as músicas com gênero principal Romântico.

Portanto, foi mapeada de acordo com os critérios acima, a classe de cada letra musical, resultando na nova distribuição visível na figura 3.8.

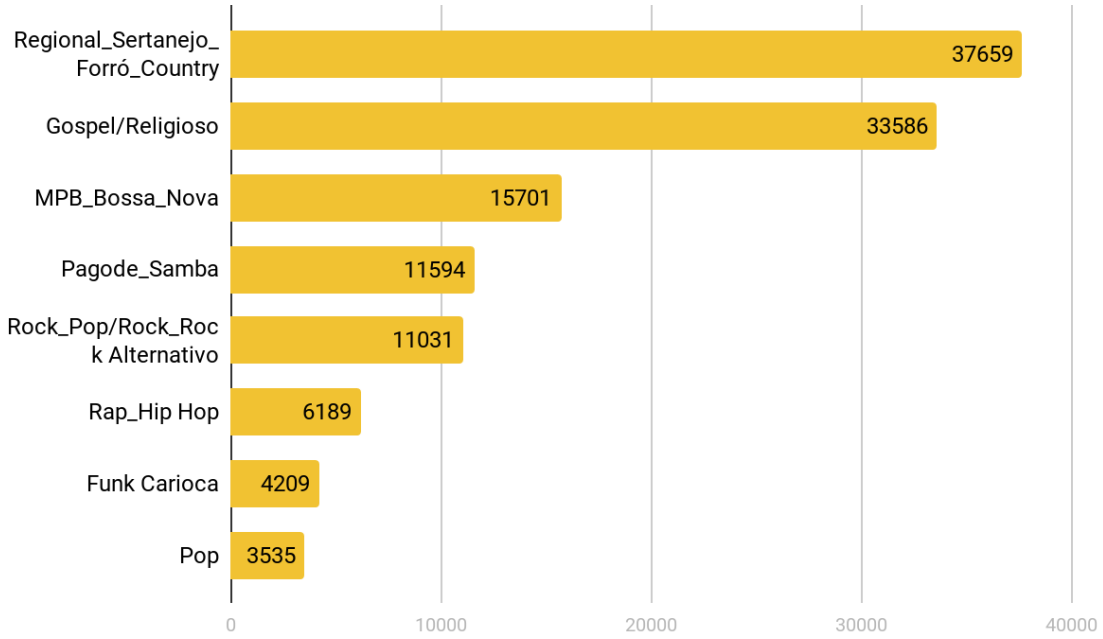


Figura 3.8 – Distribuição das classes mapeadas no dataset.

3.4 – Pré-processamento do dataset

Antes de realizar qualquer processamento de texto é necessário avaliar o conteúdo do texto em si e identificar possíveis problemas que podem impactar no desempenho dos modelos. As letras musicais do Vagalume são, por diversas vezes, cadastradas pela comunidade que utiliza a plataforma, e por isso algumas situações não desejáveis ocorrem nos dados.

Para prevenir problemas utilizando o dataset foi necessário:

- Remover tags que se encontravam dentro das letras musicais, pois muitas letras possuem tags do tipo: [Nome do cantor], [Refrão], (2x), etc. Sendo assim, todo o conteúdo entre chaves ou parêntesis foi removido das letras. Isto se fez necessário uma vez que o classificador poderia receber pistas destas tags, como por exemplo o nome do cantor e associar diretamente o gênero correto em vista disto;
- Remover conteúdo entre hífens, sinais de igual ou asteriscos, tal como "----Refrão-----". Porque além de não contribuírem para a predição ainda não fazem parte de fato da letra musical;
- Remover pontuação e demais caracteres especiais;
- Remover conteúdo após "enviado por" ou "adicionado por", já que muitos usuários da comunidade deixam sua marca na própria letra musical adicionando uma nota de rodapé com o sufixo em questão;
- Remover letras musicais que na verdade eram partituras ou instrumentais.

Também foi necessário remover a pontuação e converter as letras musicais para minúsculo. Pois posteriormente cada termo de cada letra será mapeado em um vetor e a presença de pontuação ou caracteres maiúsculos poderia atrapalhar este mapeamento.

Capítulo 4

Utilização do PyText

4.1 – Estrutura geral dos testes realizados

O PyText disponibiliza para classificação de texto dois modelos preditivos: (1) Modelo CNN; (2) Modelo BiLSTM com atenção; foram realizados testes com ambos os modelos em diferentes configurações.

Para a camada de *embedding* foram utilizados vocabulário externos, adquiridos no Núcleo Interinstitucional de Linguística Computacional da USP, este por sua vez disponibiliza vocabulários para diferentes dimensionalidades de saída utilizando diferentes técnicas (fastText, Glove, etc) [40]. Foram realizados testes com o conjunto fastText - CBOW e Glove utilizando 300 dimensões, no qual o Glove obteve melhores resultados, e, assim, foi mantido para a classificação dos modelos.

O *dataset* resultante da etapa 3.5 continha aproximadamente 123 mil letras musicais. Dele foram extraídos dois datasets, um de treino e um de validação. O de treino contava com 80% das letras e o de teste o restante.

Muitas das configurações foram mantidas no padrão do PyText, como a função de custo, que foi mantida a função de entropia cruzada, e o tamanho do *batch*, que foi mantido 16. Cada modelo treinou em até 15 *epochs*, todavia eles sempre paravam antes disso pois foi configurado *early_stop* como 2, ou seja, se não houvesse melhoria de performance no conjunto de teste em 2 *epochs* o treinamento terminaria. Também foram testados alguns valores de *dropout* para ambos os modelos, e foi deixado o que obteve melhor performance final.

Para amenizar o problema do alto desbalanceamento entre as classes tentou-se atribuir pesos diferentes para cada classe na função de custo, de forma que as classes mais raras obtivessem maior peso, entretanto a forma de realizar tal procedimento não é bem documentada no PyText. Foram realizadas algumas tentativas de se definir os pesos, porém nenhuma funcionou adequadamente, o que acabou por penalizar os resultados das classes minoritárias.

Os treinamentos foram realizados em uma máquina com processador Intel Core i5 3450 4GHz, memória 8Gb DDR3 1600Mhz e uma NVidia GTX 970, e levaram em média 2h para terminar.

4.2 – Modelo BiLSTM com atenção

Para o modelo BiLSTM foi mantida a configuração padrão para o número de camadas, pois não houve melhoria após adicionar mais camadas, ou seja, foi utilizada uma camada com 32 neurônios para cada rede LSTM e a camada de atenção utilizada foi mantida com 64 neurônios. Na figura 4.1 é possível visualizar o JSON de configuração usado para o modelo.

```
{
  "task": {
   "DocumentClassificationTask": {
     "data": {
       "source": {
         "TSVDataSource": {
           "field_names": ["label", "text"],
           "train_filename": "data/train_data.tsv",
           "test_filename": "data/test_data.tsv",
           "eval_filename": "data/test_data.tsv"
         }
       }
     },
     "model": {
       "DocModel": {
         "representation": {
           "BiLSTMDocAttention": {
             "dropout": 0.8,
             "lstm": {
               "dropout": 0.4,
               "lstm_dim": 32,
               "num_layers": 1,
               "bidirectional": true
             }
           }
         },
         "embedding": {
           "embed_dim": 300,
           "embedding_init_strategy": "zero",
           "export_input_names": [
```

```
                "tokens_vals"
            ],
            "pretrained_embeddings_path": "glove_s300.txt"
          },
          "output_layer": {
            "loss": {
                "CrossEntropyLoss": {}
            },
            "label_weights": {
              "Regional_Sertanejo_Forró_Country": 3.2,
              "Gospel/Religioso": 3.6,
              "MPB_Bossa_Nova": 7.8,
              "Pagode_Samba": 10.0,
              "Rock_Pop/Rock_Rock_Alternativo": 11.0,
              "Rap_Hip_Hop": 20.0,
              "Funk_Carioca": 29.0,
              "Pop": 35.0
            }
          }
        },
      },
      "trainer": {
        "epochs": 15,
        "early_stop_after": 2
      },
      "metric_reporter": {
        "output_path": "test_out.txt",
        "model_select_metric": "accuracy",
        "target_label": null,
        "text_column_names": [
            "text"
        ]
      }
    }
  },
  "export_torchscript_path": "/tmp/new_docnn.pt1"
}
```

Figura 4.1 – JSON de configuração do modelo BiLSTM.

4.3 – Modelo CNN

Para o modelo CNN também foram utilizados os parâmetros padrões, pois não houve melhora com a alteração e acréscimo de *kernels* ou camadas. Logo, foram mantidos 100 *kernels* de tamanho 3 e 4.

Na figura 4.2 é possível visualizar o *representation* utilizado para o modelo. O restante das configurações são iguais às exibidas na figura 4.1.

```
"representation": {
           "DocNNRepresentation": {
             "dropout": 0.6,
             "cnn": {
               "kernel_num": 100,
               "kernel_sizes": [
                   3,
                   4
               ]
             }
           }
         }
```

Figura 4.2 – Representation do JSON de configuração do modelo CNN.

4.4 – Resultados

Na figura 4.3 é possível visualizar os resultados de ambas as redes de forma comparativa. É perceptível que ambas obtiveram resultados semelhantes, com uma pequena vantagem para a rede BiLSTM. Pelo número de classes é possível inferir que as redes conseguiram obter bom desempenho, possivelmente só não é ainda melhor pelo problema do correto mapeamento de gêneros para cada letra.

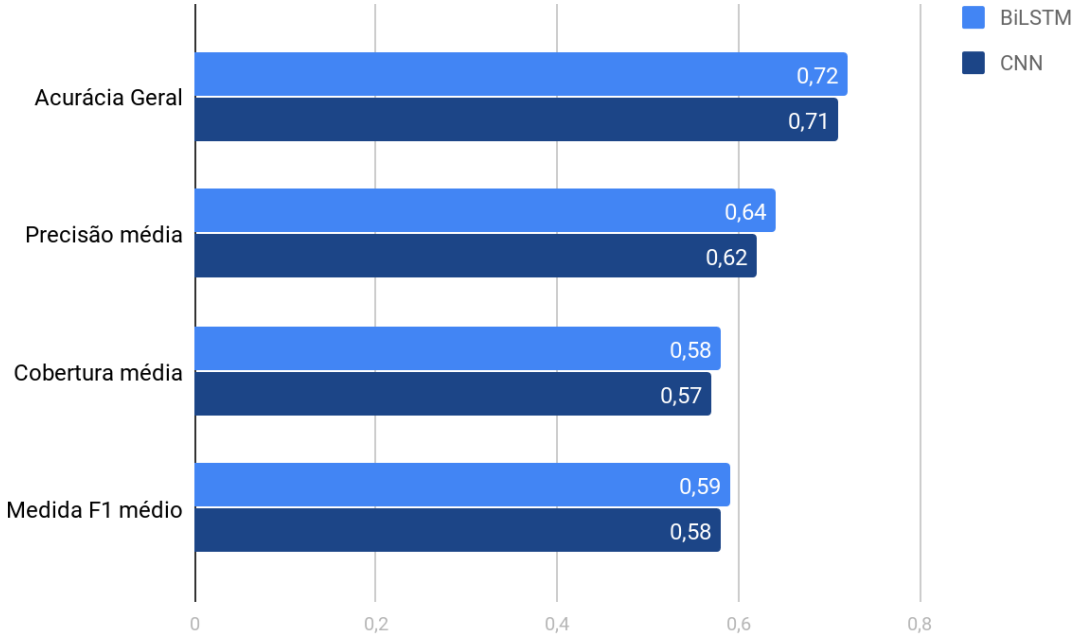


Figura 4.3 – Resultados das redes BiLTSM e CNN nos dados do dataset.

Na tabela 4.1 estão disponíveis os resultados por classe para cada modelo. Novamente as redes não apresentam grandes diferenças nos resultados. É perceptível que as classes minoritárias não obtiveram altas taxas de precisão e cobertura, devido ao alto desbalanceamento do dataset. No anexo A são apresentadas as matrizes de confusão geradas por ambas as redes.

Tabela 4.1 – Resultados das redes CNN e BiLSTM no dataset de letras musicais.

Classe	Precisão	Cobertura	Medida F1
	Rede CNN		
Gospel/Religioso	0.87	0.93	0.90
Regional_Sertanejo_Forró_Country	0.70	0.85	0.77
Rap_Hip_Hop	0.78	0.82	0.80
Funk_Carioca	0.74	0.60	0.67
MPB_Bossa_Nova	0.54	0.54	0.54
Rock_Pop/Rock_Rock_Alternativo	0.50	0.46	0.48
Pagode_Samba	0.58	0.32	0.42
Pop	0.25	0.02	0.04
	Rede BiLSTM com atenção		
Gospel/Religioso	0.89	0.94	0.91
Regional_Sertanejo_Forró_Country	0.71	0.87	0.78
Rap_Hip_Hop	0.87	0.76	0.81
Funk_Carioca	0.77	0.60	0.68
MPB_Bossa_Nova	0.58	0.52	0.55
Rock_Pop/Rock_Rock_Alternativo	0.52	0.50	0.51
Pagode_Samba	0.57	0.43	0.49
Pop	0.24	0.01	0.01

Capítulo 5

Considerações finais

5.1 – Conclusões

Este trabalho demonstrou com sucesso o uso da nova ferramenta de processamento de texto PyText para uma aplicação de classificação de texto livre. Foram retratados todos os passos para a execução de dois modelos disponíveis na ferramenta, desde a instalação até a validação dos resultados, também foi apresentada a arquitetura macroscópica da ferramenta. Para possibilitar tal estudo foi construído um dataset de letras musicais a partir do portal Vagalume [1].

Olhando em retrospecto, fica claro que o framework PyText é de fato adequado para a produção de novas aplicações de processamento texto utilizando arquiteturas já conhecidas. O desenvolvimento das aplicações é simples, basta alterar os parâmetros corretos no arquivo JSON de configuração especificado, ou seja, não é necessário produzir um modelo do zero para toda nova aplicação. Entretanto não é claro se o framework seria adequado no desenvolvimento de novos modelos com arquiteturas diferentes das previamente suportadas.

Dentre as barreiras para o uso do framework podemos destacar a falta de uma documentação mais detalhada sobre as diversas funcionalidades disponíveis, o número limitado de tutoriais e a comunidade da ferramenta que ainda é bem pequena, entretanto podemos esperar melhoras nesses pontos conforme o projeto adquire maturidade. Além disso também há a grande dificuldade em se instalar os drivers específicos para utilizar as funcionalidades CUDA do PyTorch.

5.2 – Trabalhos futuros

Para trabalhos futuros pode-se pensar em: (1) verificar o desempenho da aplicação Caffe2 para a qual o PyText exporta os modelos prontos; (2) testar os demais modelos

presentes no PyText, como aqueles direcionados para identificação de entidades; (3) comparar as funcionalidades do framework com as de seus concorrentes como AllenNLP e CoreNLP; (4) detalhar a estrutura interna do framework de forma mais profunda; (5) repetir o experimento utilizando *undersampling* randômico; (6) repetir o experimento fazendo de uso de uma rede GAN para produção de novas instâncias de treino para as classes minoritárias; (7) repetir o experimento com diferentes pesos na função de custo, de forma que as classes minoritárias possuam peso maior.

Anexo A

Matrizes de confusão

Para tornar melhor a visualização foram utilizados apelidos para as classes nas matrizes de confusão. A tabela com o mapeamento se encontra abaixo e as matrizes vem imediatamente após.

Classe	**Apelido**
Funk_Carioca	Funk
Gospel/Religioso	Gospel
MPB_Bossa_Nova	MPB
Pagode_Samba	Samba
Pop	Pop
Rap_Hip_Hop	Rap
Regional_Sertanejo_Forró_Country	Forró
Rock_Pop/Rock_Rock_Alternativo	Rock

Cada linha da matriz representa o resultado do classificador para aquela classe no conjunto de teste utilizado. Por exemplo: pode-se perceber que a rede CNN mapeou 14 letras na classe Pop que de fato eram da mesma classe, entretanto também mapeou erroneamente 230 letras de Pop na classe Forró.

Rede CNN								
	Funk	Gospel	MPB	Samba	Pop	Rap	Forró	Rock
Funk	496	24	16	40	4	69	145	28
Gospel	4	6159	86	15	3	38	206	99
MPB	11	122	1725	246	3	43	703	365
Samba	25	72	417	747	9	21	930	86
Pop	19	103	106	21	14	32	230	171
Rap	31	34	17	16	5	987	42	73
Forró	74	307	397	147	13	32	6382	188
Rock	9	222	440	47	6	43	446	1027

Rede BiLSTM com atenção								
	Funk	Gospel	MPB	Samba	Pop	Rap	Forró	Rock
Funk	495	21	13	39	3	38	175	38
Gospel	5	6207	64	32	1	25	198	78
MPB	9	91	1685	348	0	19	686	380
Samba	21	59	350	986	4	9	805	73
Pop	24	95	112	39	4	21	221	180
Rap	26	50	22	21	0	913	79	94
Forró	53	233	313	205	2	6	6553	175
Rock	11	227	347	56	3	23	460	1113

Referências bibliográficas

[1] “Vagalume”, https://www.vagalume.com.br/, 2018, (Acesso em 16 Junho 2019).

[2] “Artificial Intelligence in the Rising Wave of Deep Learning: The Historical Path and Future Outlook”, https://ieeexplore.ieee.org/document/8253597/, 2018, (Acesso em 21 Junho 2019).

[3] “Seleção de Gênero Músical”, https://github.com/vinicius-alves/Selecao-Genero-Musical, 2019, (Acesso em 21 Junho 2019).

[4] “scikit-learn”, https://scikit-learn.org (Acesso em 27 Agosto 2019)

[5] “PyTorch”, https://pytorch.org/ (Acesso em 27 Agosto 2019)

[6] “TensorFlow”, https://www.tensorflow.org/ (Acesso em 27 Agosto 2019)

[7] “Compare two deep learning frameworks: TensorFlow and Pytorch”, https://developer.ibm.com/blogs/two-famous-deep-learning-frameworks-compared-tensorflow-vs-pytorch/, 2018, (Acesso em 21 Junho 2019).

[8] “TensorFlow Vs PyTorch: Top 10 Differences Between The Two ML Libraries”, https://www.analyticsindiamag.com/tensorflow-vs-pytorch-top-10-differences-between-the-two-ml-libraries/, 2019, (Acesso em 21 Junho 2019).

[9] “9 Reasons Why PyTorch Will Become Your Favourite Deep Learning Tool”, https://www.analyticsindiamag.com/9-reasons-why-pytorch-will-become-your-favourite-deep-learning-tool/, 2018, (Acesso em 21 Junho 2019).

[10] AHMED, A., LAKHOTIA, K., ZHAO, S., MOHIT, M., BARLAS, O., ARORA, A., GUPTA, S., DEWAN, C., NELSON-LINDALL, Steff, RUSHIN, S., “PyText: A seamless path from NLP research to production”, https://research.fb.com/publications/pytext-a-seamless-path-from-nlp-research-to-production/, 2018, (Acesso em 16 Junho 2019).

[11] “PyText Documentation”, https://pytext.readthedocs.io/, 2018, (Acesso em 21 Junho 2019).

[12] “PyText Users”, https://www.facebook.com/groups/pytext/, 2018, (Acesso em 21 Junho 2019).

[13] MIKOLOV, T., LE, Q. V., SUTSKEVER, I., “Exploiting Similarities among Languages for Machine Translation”, https://arxiv.org/pdf/1309.4168.pdf, 2013, (Acesso em 21 Junho 2019).

[14] “FastText”, https://fasttext.cc/, 2019, (Acesso em 21 Junho 2019).

[15] PENNINGTON, J., SOCHER, R., MANNING, C. D., " GloVe: Global Vectors for Word Representation" https://nlp.stanford.edu/projects/glove/, 2014, (Acesso em 21 Junho 2019).

[16]"Imbalanced-Learn,under-sampling",https://imbalanced-learn.readthedocs.io/en/stable/under_sampling.html, 2017, (Acesso em 23 Junho 2019).

[17] LIU, A., "The Effect of Oversampling and Undersampling on Classifying Imbalanced Text Datasets", http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.5878&rep=rep1&type=pdf, 2004, (Acesso em 23 Junho 2019).

[18] DOUZAS, G., BACAO, F., "Expert Systems with Applications", pp. 464 –471,, https://www.sciencedirect.com/science/article/pii/S0957417417306346, 2018 (Acesso em 23 Junho 2019).

[19] RUBINSTEIN, R. Y., KROESE, D. P., "The Cross-Entropy Method."New York: Springer, 2004. Disponível em: https://books.google.com.br/books?id=8KgACAAAQBAJ&lpg=PA1&ots=BcpsIamE_9, (Acesso em 21 Junho 2019)

[20] MIKOLOV, T. et al., "Recurrent Neural Network Based Language Model", In: Interspeech, pp. 1045–1048, Japan, 2010. Disponível em: https://www.isca-speech.org/archive/archive_papers/interspeech_2010/i10_1045.pdf (Acesso em 21 Junho 2019).

[21] GERS, F. A., SCHMIDHUBER, J., CUMMINS, F., " Learning to forget: continual prediction with LSTM", In: Ninth International Conference on Artificial Neural Networks, Edinburgh, 1999. Disponível em: https://ieeexplore.ieee.org/document/818041 (Acesso em 21 Junho 2019).

[22] ZHANG, Y., WANG, J., ZHANG, X., "YNU-HPCC at SemEval-2018 Task 1: BiLSTM with Attention BasedSentiment Analysis for Affect in Tweets", In: 12th International Workshop on Semantic Evaluation, pp. 273 – 278, Louisiana, 2018. Disponível em: https://www.aclweb.org/anthology/S18-1040 (Acesso em 21 Junho 2019).

[23] RUSSAKOVSKY, O. et al., "ImageNet Large Scale Visual Recognition Challenge", https://arxiv.org/pdf/1409.0575.pdf, 2015, (Acesso em 21 Junho 2019).

[24] KIM, Y., "Convolutional Neural Networks for Sentence Classification", https://arxiv.org/pdf/1408.5882.pdf, 2014, (Acesso em 21 Junho 2019).

[25] "Max-pooling/Pooling", https://computersciencewiki.org/index.php/Max-pooling_/_Pooling, 2018, (Acesso em 21 Junho 2019).

[26] GIBSON,A., PATTERSON,J., "Deep Learning", https://www.oreilly.com/library/view/deep-learning/9781491924570/, 2017, (Acesso em 21 Junho 2019).

[27] GEROSA,L., “Consumer Business Complaints in Brazil”, https://www.kaggle.com/gerosa/procon, 2017, (Acesso em 20 Julho 2019).

[28] “Wikipedia page in brazilian”, portuguese https://www.kaggle.com/tiarles/wiki-portugues, 2018, (Acesso em 20 Julho 2019).

[29] “Brazilian E-Commerce Public Dataset by Olist”, https://www.kaggle.com/olistbr/brazilian-ecommerce, 2018, (Acesso em 20 Julho 2019).

[30] “Fake.Br Corpus”, https://github.com/roneysco/Fake.br-Corpus, 2019, (Acesso em 20 Julho 2019).

[31] “Tweets_eleicao2018”, https://www.kaggle.com/natanaelsilva/tweets-eleicao2018, 2018, (Acesso em 20 Julho 2019).

[32] NEISSE, A., “Song lyrics from 6 musical genres: Data scraped from the website “vagalume.com.br”, https://www.kaggle.com/neisse/scrapped-lyrics-from-6-genres#lyrics-data.csv, 2019, (Acesso em 20 Julho 2019).

[33] “MilkQA Dataset”, http://www.nilc.icmc.usp.br/nilc/index.php/milkqa/, 2017, (Acesso em 20 Julho 2019).

[34] “LeNER-Br: a Dataset for Named Entity Recognition in Brazilian Legal Text”, https://cic.unb.br/~teodecampos/LeNER-Br/, 2018, (Acesso em 20 Julho 2019).

[35] NEISSE,A., “ Scraping lyrics from Vagalume” https://aneisse.com/post/2019-02-10-music-data-scraping/2019-02-10-music-data-scraping/, 2019, (Acesso em 20 Julho 2019).

[36] “Langdetect”,https://github.com/Mimino666/langdetect, (Acesso em 20 Julho 2019).

[37] “TextBlob”,https://github.com/sloria/TextBlob, (Acesso em 20 Julho 2019).

[38] “Documentação da API Cloud Translation”, https://cloud.google.com/translate/docs/, (Acesso em 20 Julho 2019).

[39] “Polyglot”, https://github.com/aboSamoor/polyglot, (Acesso em 20 Julho 2019).

[40] “Repositório de Word Embeddings do Núcleo Interinstitucional de Linguística Computacional (NILC)”, http://nilc.icmc.usp.br/embeddings, 2017, (Acesso em 21 Junho 2019).