PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Inteligência Artificial e Aprendizado de Máquina

Johanes Severo dos Santos Venâncio

Classificação de Itens do Exame Nacional do Ensino Médio (Enem)

Brasília
2023

Johanes Severo dos Santos Venâncio

CLASSIFICAÇÃO DE ITENS DO EXAME NACIONAL DO ENSINO MÉDIO (ENEM)

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Inteligência Artificial e Aprendizado de Máquina, como requisito parcial à obtenção do título de *Especialista*.

Belo Horizonte
Abril de 2023

SUMÁRIO

1. Introdução

O Exame Nacional do Ensino Médio (Enem) utiliza a Teoria de Resposta ao Item (TRI) na correção das questões. Essa teoria considera que a probabilidade de um indivíduo acertar ou errar uma questão depende não só de seu conhecimento sobre o conteúdo da questão, mas também das características da questão em si, como seu grau de dificuldade e discriminação. Dessa forma, a TRI é capaz de avaliar o nível de proficiência dos participantes de forma mais precisa e justa, levando em consideração as particularidades de cada item da prova.

Na Teoria de Resposta ao Item (TRI) possui três parâmetros que são o parâmetro de dificuldade (a), o parâmetro de discriminação (b) e o parâmetro de acerto ao acaso (c).

O parâmetro "a" é chamado de dificuldade do item e representa quanto a proficiência é necessária para responder corretamente ao item. Um item com um valor alto de a é considerado mais difícil de ser respondido corretamente, enquanto um item com um valor baixo de a é considerado mais fácil.

O parâmetro "b" é chamado de discriminação do item se refere à capacidade de um item em diferenciar os indivíduos que possuem dominância na habilidade avaliada daqueles que não a possuem. Em outras palavras, quanto maior o valor do parâmetro de discriminação, mais capaz o item será de identificar corretamente os indivíduos que possuem a habilidade e os que não a possuem.

O parâmetro de acerto ao acaso “c” é a probabilidade de um indivíduo que não possui a habilidade exigida pelo item acertá-lo por sorte ou pelo processo de tentativa e erro. Em outras palavras, o parâmetro de acerto ao acaso representa a chance de um indivíduo sem a habilidade necessária acertar o item por acidente.

Os valores de a e b são geralmente estimados a partir dos dados usando técnicas de ajuste de curva, como o método dos mínimos quadrados. Uma vez estimados, esses parâmetros podem ser usados para calcular a proficiência de indivíduos ou grupos de indivíduos que responderam ao item.

Na TRI θ (theta) é a variável de proficiência, é uma medida da habilidade de um indivíduo em relação a um determinado item ou conjunto de itens. Na Teoria de Resposta ao Item, a proficiência é geralmente representada como uma escala numérica contínua, onde valores mais altos de θ indicam maior habilidade e valores mais baixos de θ indicam menor

habilidade. A proficiência é usada como uma variável independente na função de probabilidade para estimar a probabilidade de um indivíduo responder corretamente a um item dado sua habilidade. Em geral, essa teoria usa uma função de probabilidade para modelar a relação entre a proficiência de um indivíduo (θ) e a chance de ele responder corretamente a um item (p).

Para calcular θ, é necessário estimar os parâmetros a e b a partir das respostas de um grupo de indivíduos a um conjunto de itens. Isso pode ser feito usando um método de otimização bem como o método dos mínimos quadrados. Em seguida, θ pode ser calculado usando a função inversa da função logística, ou usando uma técnica de estimação de máxima verossimilhança.

O pré-teste de itens é realizado no Enem para avaliar a validade dos itens antes de serem incluídos na prova final. O objetivo é verificar se os itens estão medindo o que se pretende e se eles são apropriados para a avaliação dos alunos. Isso inclui a avaliação da dificuldade dos itens, a capacidade de discriminação entre os alunos com diferentes níveis de habilidade e a precisão dos itens na avaliação da habilidade. O resultado do pré-teste é usado para selecionar os itens que serão incluídos na prova final e para ajustar a prova para garantir que ela esteja avaliando de maneira eficiente e precisa o que se pretende.

A calibração dos itens é o processo de estimar ou determinar os parâmetros de discriminação, dificuldade e acerto ao acaso de cada item em uma prova. A calibração dos itens é realizada por meio de um pré-teste, que é uma aplicação de uma amostra de participantes com o objetivo de estimar os parâmetros dos itens.

O resultado da calibração dos itens é utilizado para avaliar a qualidade dos itens, selecionar os itens para a prova final, e/ou determinar a pontuação dos participantes. Portanto, a calibração dos itens é uma etapa importante na implementação da Teoria de Resposta ao Item (TRI).

Quando um item calibrado é aplicado, é esperado que a média dos resultados dos participantes seja compatível com o valor do parâmetro de dificuldade do item. Além disso, também é esperado que a diferença entre a taxa de acerto dos participantes que dominam a habilidade avaliada e a taxa de acerto dos participantes que não a dominam seja compatível com o valor do parâmetro de discriminação do item. Assim, a calibração dos itens por meio de um pré-teste ajuda a garantir que o item seja uma medida válida e confiável da habilidade que se quer avaliar.

Se a média dos resultados dos participantes não for compatível com o valor do parâmetro de dificuldade e discriminação do item, isso pode indicar que a calibração do item não foi realizada corretamente ou que houve algum problema na aplicação do item. Neste caso, pode ser necessário realizar uma nova calibração do item ou revisar a aplicação para garantir a validade dos resultados. Além disso, pode ser necessário revisar ou ajustar a equação de estimação de proficiência utilizada para calcular os resultados.

Um item não calibrou significa que os resultados obtidos na aplicação do item não correspondem aos valores estimados pelo parâmetro de dificuldade e discriminação previamente definidos para o item. Isso pode ser devido a vários fatores, como problemas na construção do item, aplicação inadequada ou distorção na resposta dos participantes. A falta de calibração de um item pode afetar a precisão e a validade das medidas obtidas a partir desse item, o que pode prejudicar a avaliação em geral.

Nesse sentido, o presente trabalho se propõe a explorar a utilização de um modelo de *machine learning* para classificar um item elaborado para o Exame Nacional do Ensino Médio com base no texto do item e no texto das alternativas, dentro de quatro faixas de valores do parâmetro de discriminação, parâmetro de dificuldade e o parâmetro de acerto ao acaso, ou seja, pretende-se resolver um problema de classificação de texto. Esse problema se situa no processo de aprendizagem supervisionada, no campo da Inteligência Artificial. Isso pode ajudar no processo de uma avaliação educacional, como o Enem, porque permite uma seleção mais eficiente e precisa de itens para a avaliação. Essa classificação com base nas faixas de valores dos parâmetros de dificuldade, discriminação e acerto ao acaso pode permitir a identificação de itens que possuem características desejáveis, tais como uma boa discriminação entre os participantes que dominam e não dominam a habilidade avaliada, e uma dificuldade apropriada para a avaliação. O uso de um modelo de classificação de itens pode ser uma ferramenta valiosa para melhorar a qualidade e a eficiência da avaliação educacional.

O modelo de classificação de itens do Enem pode fornecer uma estimativa inicial dos parâmetros de discriminação, dificuldade e acerto ao acaso, porém, não eliminando a necessidade de realização de um pré-teste para validar a precisão desses valores. Isso porque existem muitos fatores que podem afetar o desempenho dos itens, como a clareza e a concisão da pergunta, o nível de habilidade dos participantes, entre outros. Portanto, o pré-teste permitiria ajustar e validar esses valores antes de utilizá-los para fins de avaliação.

2. Descrição do Problema e da Solução Proposta

2.1. Justificativa

Um modelo de machine learning de classificação de texto de itens de prova pode ajudar o processo de uma avaliação educacional, como o Enem, pois pode proporcionar uma seleção mais eficiente e precisa de itens para a avaliação. Essa classificação do item nas faixas de valores dos parâmetros de dificuldade, discriminação e acerto ao acaso pode permitir a identificação de itens que possuem características desejáveis, tais como uma boa discriminação entre os participantes que dominam e não dominam a habilidade avaliada, e uma dificuldade apropriada para a avaliação. Assim, a justificativa para a realização do trabalho é que o uso de um modelo de classificação de texto de itens pode ser uma ferramenta valiosa para melhorar a qualidade e a eficiência da avaliação educacional. Esse modelo pode fornecer uma estimativa inicial dos parâmetros de discriminação, dificuldade e acerto ao acaso, mesmo não eliminando a necessidade de realização de um pré-teste para validar a precisão desses valores. Isso porque existem muitos fatores que podem afetar o desempenho dos itens, como a clareza e a concisão da pergunta, o nível de habilidade dos participantes, entre outros. Portanto, o pré-teste permite ajustar e validar esses valores antes de utilizá-los para fins de avaliação.

2.2. Objetivo

O objetivo do presente trabalho é obter um modelo de machine learning para classificação do item com base no texto do texto-base e alternativas em uma das quatro faixas de valores dos parâmetros dificuldade, discriminação e acerto ao acaso determinadas. Esse objetivo terá as seguintes etapas:

- Coleta de dados: será coletado o itens das provas do Enem e os parâmetros dos itens com base nos microdados publicados pelo Instituto Nacional de Pesquisas Educacionais AnísioTeixeira (Inep);
- Processamento/Tratamento de dados: a base com o texto do itens e as classes anotadas será montada para ser utilizada na preparação de dados para o modelo;
- Análise e exploração dos dados: será feita uma exploração dos dados presentes na base montada;
- Preparação dos dados para os Modelos de Aprendizado de Máquina: os dados serão preparados para o modelo a partir da base montada;
- Aplicação de modelos de Aprendizado de Máquina: o modelo será treinado utilizando uma Rede Neural Convolucional para classificação de texto;

- Avaliação dos modelos de Aprendizado de Máquina e Discussão dos Resultados: as métricas de avaliação do modelo serão apresentadas.

2.3. Solução proposta

A solução proposta foi treinar uma Rede Neural Convolucional para classificação do texto do texto-base dos itens e das alternativas. A rede neural classifica os textos em quatro faixas determinadas dos parâmetros dos itens.

2.4. Links para os códigos fonte:

Link para o projeto (entrega A1) e os dados: JOHANES_SEVERO_PROJETO_INTEGRADO_ENTREGA_A1.zip

3. Coleta de Dados

Os dados foram obtidos a partir dos microdados do Enem disponíveis em: https://www.gov.br/inep/pt-br/acesso-a-informacao/dados-abertos/microdados/enem.
Utilizou-se as provas em ".pdf" disponíveis no diretório "PROVAS E GABARITOS". Alguns arquivos ".pdf" eram convertidos para texto e ficavam ilegíveis então, algumas provas foram extraídas do endereço: http://educacao.globo.com/provas/enem-<ano>/. Os parâmetros e outros dados sobre os itens estão disponíveis no diretório "DADOS" no arquivo "ITENS_PROVA_<ano>.csv". Os parâmetros dos itens estão divulgados a partir do ano de 2009 então, somente foi possível utilizar os dados a partir desses anos para treinar o modelo. O processo de coleta dos textos foi basicamente este:

- Extrair o texto dos itens dos arquivos ".pdf" ou da página da web e adicionar os textos em um arquivo ".csv" junto com algumas características conhecidas;
- Concatenar os textos dos itens, utilizando as características conhecidas, com o arquivo de microdados de itens gerando um arquivo ".csv" com os textos dos itens e os parâmetros em um único arquivo ".csv";
- Após obtido todos os arquivos de todos os anos com a mesma estrutura, foi feito o append desses arquivos em um único arquivo ".csv" que foi utilizada no pré-processamento e tratamento de dados para montar a base utilizada no treinamento do modelo;

3.1. Extração do texto dos itens

Quando foi necessário extrair os textos dos itens do arquivo "PDF" da prova, primeiro o arquivo foi convertido em texto por meio da biblioteca PyPDF2 baseada em Python e, em seguida, salvo em um arquivo ".txt". Em seguida, o arquivo ".txt" foi lido e alguns textos foram removidos utilizando expressões regulares antes de extrair os textos das questões e alternativas. Para localizar cada questão, foi utilizado um padrão específico de expressão regular. Como os arquivos "PDF" de cada ano tinham particularidades específicas, optou-se por criar funções de extração para cada ano, aperfeiçoando-as a cada ano. Na maioria dos anos, havia vários tipos de prova diferenciados pela cor, sendo necessário correspondê-los aos microdados no arquivo correspondente, utilizando informações como posição do item (CO_POSICAO), cor da prova, área (SG_AREA) e código da prova (CO_PROVA). Para descobrir o código da prova referente ao arquivo ".pdf", foi realizada uma análise qualitativa dos dados disponíveis. Mesmo quando havia questões em comum entre os tipos de prova, optou-se por extrair de todos os tipos disponíveis para aproveitar ao máximo as questões. Os arquivos ".pdf" não extraídos foram aqueles que não puderam ser convertidos em ".txt", nos quais não foi possível identificar os parâmetros dos itens e/ou nos quais a prova não estava disponível em alguma página da web. O arquivo com o código utilizado na extração está no notebook “coleta-dados.ipynb”

Quando os textos dos itens foram extraídos a partir de uma página da web, foram utilizadas as bibliotecas Requests, BeautifulSoup e expressões regulares para localizar e extrair os textos das questões.

Foi extraído somente questões em língua portuguesa. Ao final da extração o seguinte arquivo “.csv” foi montado:

Nome do Atributo	Descrição	Tipo
co_posicao	Posição do Item na Prova	Discreta
sg_area	Área de Conhecimento do Item	Categórico nominal
co_item	Código do Item	Discreta
tx_gabarito	Gabarito do Item	Categórico nominal
co_habilidade	Habilidade do Item	Categórico nominal
in_item_aban	Indicador de item abandonado	Categórico nominal

tx_motivo_aban	Motivo para o abandono do item	Texto
nu_param_a	Parâmetro de discriminação: é o poder de discriminação do item para diferenciar os participantes que dominam dos participantes que não dominam a habilidade avaliada.	Contínua
nu_param_b	Parâmetro de dificuldade: associado à dificuldade do item, sendo que quanto maior seu valor, mais difícil é o item.	Contínua
nu_param_c	Parâmetro de acerto ao acaso: é a probabilidade de um participante acertar o item não dominando a habilidade exigida.	Contínua
tx_cor	Cor da prova	Texto
co_prova	Identificador da Prova	Discreta
id	Concatenação de co_posicao, área e cor	Categórica nominal
meio_aplicacao	Meio de aplicação da prova se digital ou em papel	Categórica nominal
texto_questao	Texto da questão incluindo as alternativas	Texto
texto_base	Texto do texto-base	Texto
alternativa_a	Texto da alternativa a	Texto
alternativa_b	Texto da alternativa b	Texto
alternativa_c	Texto da alternativa c	Texto
alternativa_d	Texto da alternativa d	Texto
alternativa_e	Texto da alternativa e	Texto

Alguns textos de alternativa não se conseguiu extrair isoladamente, então algumas linhas estão em branco. Os anos que se conseguiu extrair dados de forma automática foram: 2010, 2011, 2012, 2013, 2014, 2015, 2017, 2019, 2020 e 2021.

A quantidade de itens que se conseguiu extrair e corresponder com os parâmetros presentes nos microdados, nos arquivos “ITENS_PROVA_<ano>.csv”, foi de 1922 itens, ou seja, os dados desses 1922 itens podem ser utilizados para o pré-processamento e para o treinamento do modelo de *machine learning*. O nome do arquivo com os 1922 itens é “df_itens_geral_validos.csv”.

4. Análise e Exploração dos dados

4.1. parâmetros dos itens

A análise exploratória das amostras de parâmetros seguiu as seguintes etapas:

- Verificar a distribuição dos dados e a normalidade realizando o teste de hipótese de Shapiro das amostras dos parâmetros a, b e c sem a remoção de outliers;
    - Por exemplo, trechos de código para o parâmetro a:
        - plotar a distribuição:

```
# distribuição estatística da coluna do parâmetro a
sns.kdeplot(data=data, x="nu_param_a")
```

        - teste de shapiro:

```
# teste de normalidade de Shapiro-Wilk
stat, p = shapiro(data.nu_param_a.values)

# resultados do teste
print('Estatística do teste: ', stat)
print('Valor p: ', p)
if p > 0.05:
    print('Distribuição é normal')
else:
    print('Distribuição não é normal')
```

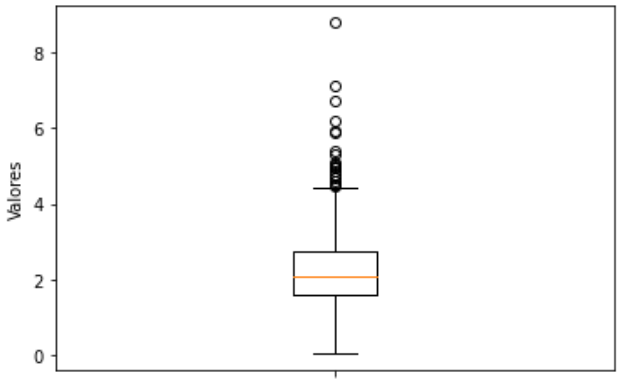
        - boxplot:

```
# boxplot dos dados
```

```
fig, ax = plt.subplots()
ax.boxplot(data.nu_param_a.values)

# Configurar eixo y
ax.set_ylabel('Valores')

# Mostrar gráfico
plt.show()
```



- Verificar a distribuição dos dados e a normalidade realizando o teste de hipótese de Shapiro das amostras dos parâmetros a, b e c com a remoção de outliers utilizando o algoritmo Local Outlier Factor (LOF);
    - Por exemplo, trechos de código para o parâmetro a:
        - Remover outliers:

```
lof = LocalOutlierFactor(n_neighbors=20)
valores =
data.nu_param_a.values.reshape(data.shape[0], 1)
# aplicar o algoritmo LOF
anomalies = lof.fit_predict(valores)

# pontuações de anomalia para cada ponto
scores = lof.negative_outlier_factor_

# um dataframe para visualizar os dados e as
pontuações de anomalia
df = pd.DataFrame(valores, columns=['valores'])
df['anomalies'] = anomalies
df['scores'] = scores
```

```
# imprimir outliers
outliers = df[df['anomalies'] == -1]
print("Outliers:")
outliers.sort_values('valores', ascending = False)
data_1 = data[~data.index.isin(df[df['anomalies'] ==
-1].index)].reset_index(drop = True)
params_a_sem_outliers = data_1.nu_param_a.values
```

- Realizar os testes de correlação não paramétrica de Spearman e Kendall para verificar a correlação dos parâmetros entre si: entre a, b, entre a, c e entre b, c;
    - Por exemplo, trechos de código para o parâmetro a:

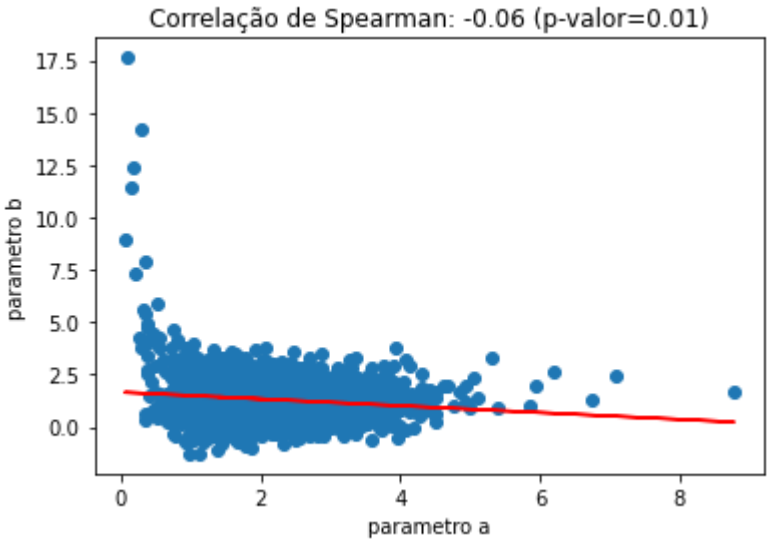
```
# linha de regressão linear
regressao = np.polyfit(data.nu_param_a,
data.nu_param_b, 1)

# correlação de Spearman
corr, p_value = spearmanr(data.nu_param_a,
data.nu_param_b)

#se p menor que 0.05 o erro é menor se a hipótese
nula for rejeitada
if p_value < 0.05:
    print("A hipótese nula foi rejeitada. Há
correlação entre as variáveis", 'p-value:', p_value,
'correlação:', corr)
else:
    print("A hipótese nula não foi rejeitada. Não há
correlação entre as variáveis.", 'p-value:',
p_value, 'correlação:', corr)

# plotar a correlação de Spearman
plt.scatter(data.nu_param_a, data.nu_param_b)
plt.plot(data.nu_param_a,
regressao[0]*data.nu_param_a + regressao[1],
color='red')
```

```
plt.title(f'Correlação de Spearman: {corr:.2f} (p-
valor={p_value:.2f})')
plt.xlabel('parametro a')
plt.ylabel('parametro b')
plt.show()
```



As conclusões obtidas foram:

- As distribuições estatísticas da amostra de parâmetros a, b e c não são normais;
- Há uma correlação ligeiramente negativa entre os parâmetros a e b, ou seja, entre os parâmetros de discriminação e dificuldade. Isso significa que quando maior a discriminação menor a dificuldade do item, mas como os valores de correlação negativa foram baixos pode ser que essa correlação seja insignificante;
- A correlação entre os parâmetros a e c é ligeiramente positiva. Uma correlação positiva entre o parâmetro de discriminação e o parâmetro de acerto ao acaso indicaria que os itens que são mais discriminativos (ou seja, que diferenciam melhor os respondentes mais habilidosos dos menos habilidosos) também são aqueles que têm uma maior probabilidade de serem acertados ao acaso por respondentes menos habilidosos. É importante destacar que a correlação positiva é baixa 0.06 (spearman) e 0.04 (Kendall);
- Não há correlação entre os parâmetros de dificuldade e de acerto ao acaso, ou seja, independente de o item ser difícil ou fácil isso não influencia o acerto ao acaso;
- Considerando os valores baixos de correlação de spearman e kendall, pode-se afirmar que para a amostra dos 1922 itens os parâmetros não se correlacionam entre si. Isso indica que não há relação monotônica entre as amostras de parâmetros entre

si e que, em geral, mudanças em um parâmetro não são acompanhadas de mudanças consistentes em outro parâmetro**. Se houvesse uma correlação muito forte entre as amostras de parâmetros bastaria treinar um modelo para um dos parâmetros, mas como não há essa correlação então o mais adequado é treinar um modelo correspondente a cada parâmetro.**

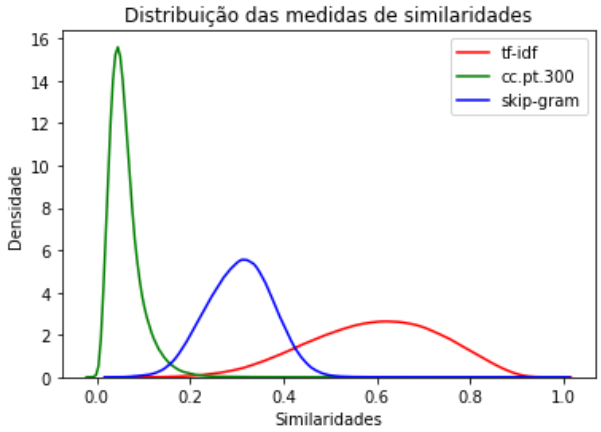
O arquivo com o código da análise exploratória dos parâmetros está no notebook "**exploracao-dados-parametros-itens.ipynb**".

4.2 Texto dos itens

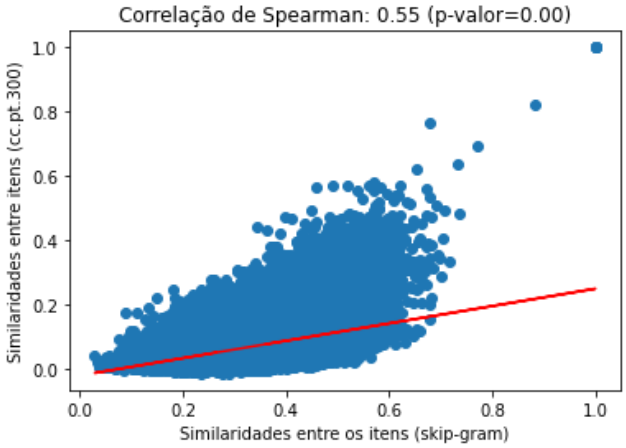
Na análise e Exploração dos dados dos textos dos itens tratou-se de criar a matriz de medidas de similaridade do cosseno item a item e verificar quais das representações está mais adequada para utilizar para o treinamento do modelo. As representações de cada questão que foram obtidas são:

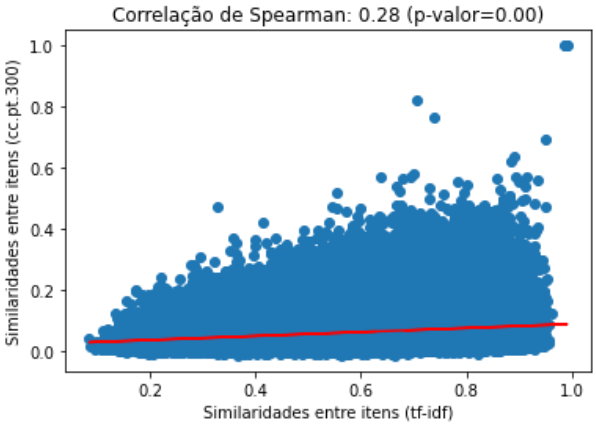
- Os vetores de word embeddings utilizando o modelo pré-treinado "cc.pt.300" da FastText;
- Os vetores de word embedding treinado somente com os textos das questões coletadas utilizando skip-gram;
- Os vetores de medidas TF-IDF referente a cada questão obtida com a coleção de questões (documentos) por área;

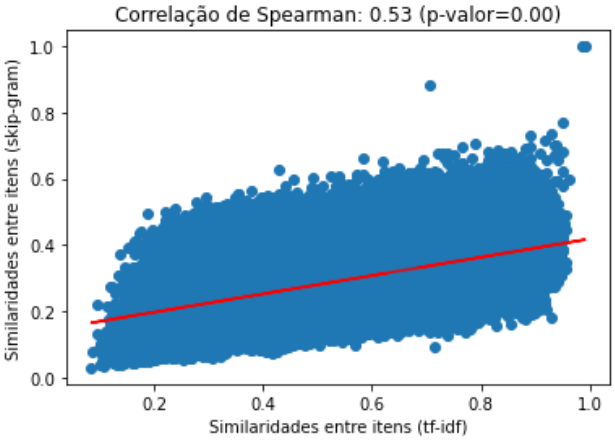
A distribuição das similaridades item a item utilizando cada uma das representações é a seguinte:



As medidas de similaridade item-item entre as três representações possuem correlações moderadas o que pode ser preciso treinar os modelos com as diversas representações.



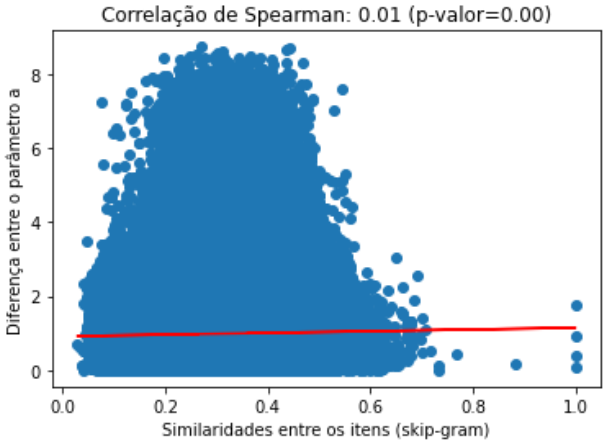




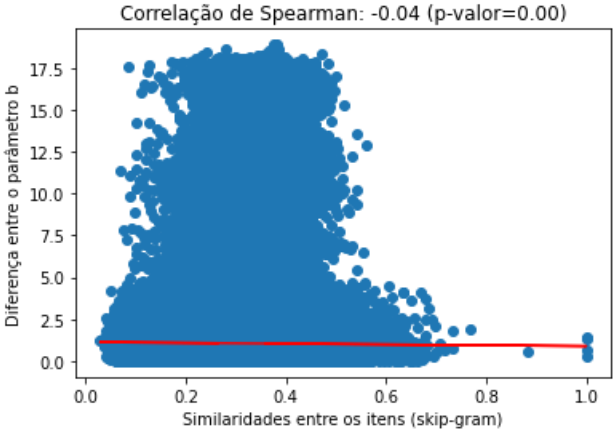
Uma hipótese inicial foi feita a de que as diferenças entre os parâmetros dos itens se correlacionam com as medidas de similaridade dos itens, ou seja, será que itens similares os semanticamente similares possuem pouca diferença em relação aos parâmetros? Espera-se que quanto maior a similaridade dos itens menor a diferença entre os parâmetros;

Verificando a correção entre similaridades (utilizando representação skip-gram) e diferença entre parâmetros observa-se:

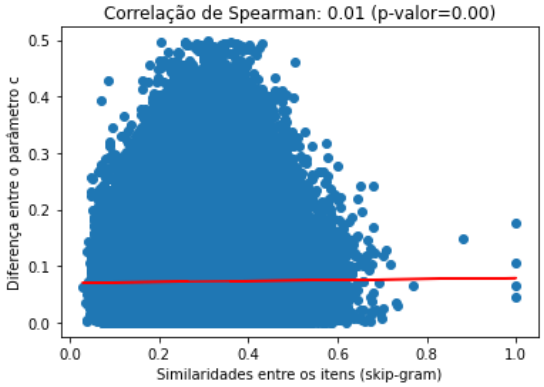
Parâmetro a:



Parâmetro b:



Parâmetro c:



Como a correlação entre diferenças e similaridade é baixa pode ser que não se consiga alta acuraria dos modelos treinados. Também se observa que a quantidade de dados é baixa, o que pode causar overffiting. É preciso destacar que mesmo com a

impressão inicial de que os modelos que serão treinados terão baixa acurácia entende-se que o objetivo do projeto é o conhecimento e a capacidade de lidar com projetos de machine learning. O arquivo com o código dessa etapa é o **“exploracao-dados-texto-questoes.ipynb”.**

5. Processamento/Tratamento de Dados

5.1. Parâmetros dos itens

O pré-processamento de dados dos parâmetros tratou-se da realização da discretização em classes de cada um do conjunto de valores de parâmetros. Assim, agruparam-se os valores contínuos em grupos distintos com o objetivo de simplificar a representação dos dados, torná-los mais fáceis de serem lidos e processados. Como os parâmetros possuem uma característica de proporção, no sentido de que, por exemplo, para o parâmetro de discriminação "a" quanto maior o valor do parâmetro de discriminação, mais capaz o item será de identificar corretamente os indivíduos que possuem a habilidade e os que não a possui, assim como os parâmetros de dificuldade e de acerto ao acaso, a melhor estratégia foi a com base em quartis. Também optou-se por isso pois a divisão em quartis é balanceada, com o mesmo número de elementos em cada classe.

Por exemplo, trecho de código a seguir discretiza a amostra dos valores do parâmetro “a” em quatro classes:

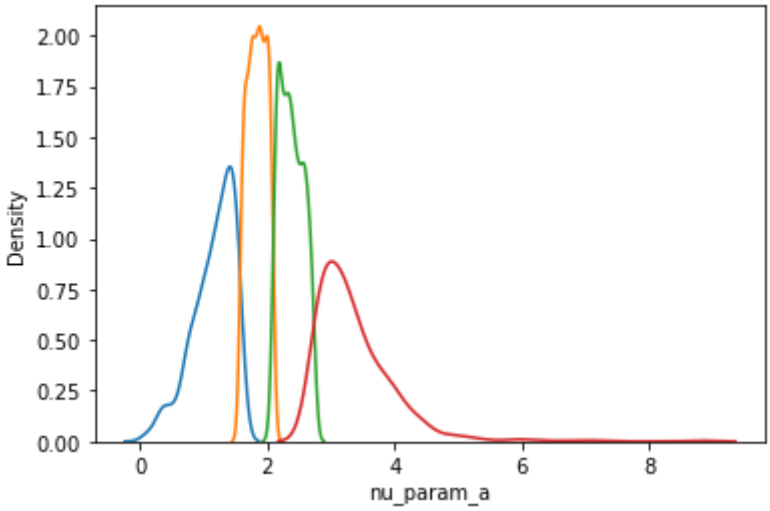
```
# definir os intervalos de discretização
intervalos = [data["nu_param_a"].min(), data["nu_param_a"].quantile(0.25), data["nu_param_a"].quantile(0.5), data["nu_param_a"].quantile(0.75), data["nu_param_a"].max()]
# criar uma nova coluna com as classes discretizadas
data["classes_param_a_1"] = pd.cut(data["nu_param_a"], intervalos, labels=['<=1.581', '>1.581 e <=2.092', '>2.092 e <=2.735', '>2.735'])
```

As classes e a quantidade de valores para cada parâmetro foram:

Parametro a	Quantidade	Parametro b	Quantidade	Parametro c	Quantidade
<=1.581	481	<=0.652	481	<=0.12487	482
>1.581 e <=2.092	480	>0.652 e <=1.224	480	>0.12487 e <=0.165	480
>2.092 e <=2.735	480	>1.224 e <=1.837	480	>0.165 e <=0.203	479

>2.735	481	>1.837	481	>0.203	481

A distribuição de valores para o parâmetro a por classe é, por exemplo, o seguinte:



O arquivo com esse processo é o notebook "**pre-processamento-dados-parametros-itens.ipynb**" e os dados tratados estão no diretório data no arquivo **"df_itens_geral_validos.pkl".**

5.2. Texto dos itens

No tratamento de dados dos textos dos itens houve as seguintes etapas:

- Tratar o texto dos itens;
- Obter os vetores de Word Embeddings dos textos dos itens;
- Obter os vetores TF-IDF dos textos dos itens;

No tratamento dos textos dos itens as seguintes ações foram feitas:

- Remover urls por meio de expressão regular;
  - `text = re.sub(r"/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i", ' ', text)`
- Transformar em minúsculas;
- Remover stop words utilizando o spacy;

    - `text_tokens = [word.lower() for word in text_tokens if word.lower() not in stop_words]`
- Lematizar utilizando o spacy;
    - `text_tokens = [nlp(word)[0].lemma_ for word in text_tokens]`
- Remover pontuações utilizando o pacote string;
    - `text_tokens = [word for word in text_tokens if word not in string.punctuation]`
- Verificar se a palavra pertence ao modelo pré-treinado “cc.pt.300”;

O "cc.pt.300" é um modelo de word embedding pré-treinado para a língua portuguesa desenvolvido pela FastText, uma biblioteca para trabalhar com processamento de texto em larga escala.

Características deste modelo incluem:

- É treinado em milhões de sentenças e documentos na língua portuguesa;
- Utiliza o algoritmo de subpalavras (subword), o que permite a representação de palavras semelhantes mesmo que não estejam presentes no treinamento;
- A dimensão do word embedding é 300, o que significa que cada palavra é representada por um vetor de 300 números;
- Fornece uma representação de palavras com similaridade sintática e semântica próxima.

Este modelo é amplamente utilizado para tarefas de processamento de texto, como análise de sentimentos, classificação de textos, clusterização de documentos, etc. Se a palavra não pertence a esse modelo significa que ela é muito estranha ou está incorreta.

Como o modelo pré-treinado “cc.pt.300" é extenso e demora a ser carregado, optou-se por armazenar os Word embeddings das palavras desse modelo em um banco de dados sqlite de forma que para obter o Word embedding de uma palavra somente é preciso pesquisar a palavra, a seguir os códigos de inserção e pesquisa:

```
#carregar o modelo
fasttext_model = gensim.models.KeyedVectors.load_word2vec_format('cc.pt.300.vec', binary=False)
```

```
# armazenar todas as palavras e embeddings do modelo pré-treinado em um banco de dados sqlite
file = os.path.join('data', 'embeddings.db')
conn = sqlite3.connect(file)

cursor = conn.cursor()

cursor.execute("""
CREATE TABLE IF NOT EXISTS embeddings (
    word TEXT PRIMARY KEY,
    embedding BLOB
)
""")

#iterar sobre as words armazenando essas words
for word in fasttext_model.index_to_key:
    cursor.execute("""
    INSERT INTO embeddings (word, embedding)
    VALUES (?,?)
    """, (word, fasttext_model[word].tobytes()))

conn.commit()
conn.close()
     # função para buscar o word embedding de uma paravra em um banco de dados sqlite
# sem precisar carregar na memória modelo de word embedding pré-treinado

def get_sqlite_we(word):
    # Conectar ao banco de dados
    file = os.path.join('data', 'embeddings.db')
    conn = sqlite3.connect(file)
    cursor = conn.cursor()

    # Executar uma consulta que seleciona a coluna com o BLOB
    cursor.execute("SELECT embedding FROM embeddings WHERE word = '"+word+"'")
```

```
# Obter o resultado da consulta
result = cursor.fetchone()

embedding = np.array([])
if result != None:
    # Converter o BLOB em array NumPy
    embedding = np.frombuffer(result[0], dtype=np.float32)

conn.commit()
conn.close()

return embedding
```

Em seguida obteve-se o vetor de Word embedding de cada questão por meio do modelo pré-treinado cc.pt.300, obteve-se os vetores de medidas TF-IDF para cada questão utilizando a classe TfidfVectorizer do sklearn e obteve-se um vetor de Word embeddings treinado com o corpus de questões coletadas utilzando o algoritmo skip-gram disponível na biblioteca gensim.

Optou-se por utilizar essas três representações para que fosse feito um comparativo de similaridades na exploração dos dados de texto para que se pudesse treinar o modelo com a representação mais adequada ou com mais de uma representação de texto.

O arquivo com esse processo é o notebook "**pre-processamento-dados-texto-questoes.ipynb**" e os dados tratados estão na pasta data no arquivo **"df_itens_geral_validos.pkl".**

6. Preparação dos Dados para os Modelos de Aprendizado de Máquina

A preparação de dados tratou-se de:

- Selecionar as colunas de interesse, no caso as representações dos textos e as classes dos parâmetros;
- Trocar os valores das classes por valores numéricos inteiros;
- Separar os dados de treino e de teste.

Utilizou-se a classe StratifiedShuffleSplit do pacote sklearn para criar uma amostra estratificada de 25% dos dados para teste e 75% dos dados para treinamento, com base nas proporções de valores das classes. É uma técnica de validação cruzada que embaralha os dados de maneira estratificada. Para cada parâmetro criou-se um conjunto de treinamento e um conjunto de teste.

O arquivo com esse processo é o notebook "**preparacao-dados-para-modelo.ipynb**" e os dados de treino e de teste estão na pasta data nos arquivos: **param_a_data_test.pkl; param_a_data_train.pkl; param_b_data_test.pkl; param_b_data_train.pkl; param_c_data_test.pkl; param_c_data_train.pkl;**

7. Aplicação de Modelos de Aprendizado de Máquina

Para o parâmetro "a" (discriminação), por exemplo, foram carregados os dados de teste "dataset_param_a_test.pkl". Foi realizado padding:

```
X = tf.keras.preprocessing.sequence.pad_sequences(X, value = 0,
padding = 'post', maxlen = max_len)
```

E também foi realizada a conversão dos word embeddings para tensor:

```
X_tensor = tf.convert_to_tensor(X, dtype=tf.float32)
```

O TensorFlow não suporta tensores com formas variadas, então todas as listas precisam ter o mesmo tamanho.

A classe que representa a rede neural convolucional é a seguinte:

```
'''
classe que representa a rede neural convolucional para
classificação de textos
'''
class DCNN(tf.keras.Model):

    '''
        tamanho_word_embbeding: tamanho do vetor de números
representando a palavra;
        qtd_filtros: número de filtros para cada dimensão;
```

```
        qtd_neuronios_camada_densa: número de neurônios da
rede neural densa;
        qtd_classes: número de classes para classificação;
        taxa_dropout: porcentagem de desativação de neurônios;
    '''
    def __init__(self,
                 tamanho_word_embbeding = 1,
                 qtd_filtros = 8,
                 qtd_neuronios_camada_densa = 64,
                 qtd_classes = 2,
                 taxa_dropout = 0.2,
                 training = False,
                 name = 'dcnn'):
        super(DCNN, self).__init__(name=name)
        #gera a matriz de embedding do vocabulário ou a
representação vetorial de cada palavra
        #camadas de convolução
        #define os filtros
        #same: retorna os mesmos dados no mesmo formato
        # para cada sentença, extrai as fetures e realiza o
treinamento das feature vector
        self.bigram    =    layers.Conv1D(filters=qtd_filtros,
kernel_size=2,    padding='same',    activation    =    'relu',
kernel_regularizer=keras.regularizers.l2(0.01))
        self.trigram    =    layers.Conv1D(filters=qtd_filtros,
kernel_size=3, padding='same', activation = 'relu')
        self.fourgram    =    layers.Conv1D(filters=qtd_filtros,
kernel_size=4, padding='same', activation = 'relu')
        self.fivegram    =    layers.Conv1D(filters=qtd_filtros,
kernel_size=5, padding='same', activation = 'relu')
        #camada max pooling
        self.pool = layers.GlobalMaxPool1D()
        #camada densa
```

```
        self.dense_1       =       layers.Dense(units       =
qtd_neuronios_camada_densa, activation = 'relu')
        self.dense_2       =       layers.Dense(units       =
qtd_neuronios_camada_densa, activation = 'relu')
        self.dense_3       =       layers.Dense(units       =
qtd_neuronios_camada_densa, activation = 'relu')
        self.dropout = layers.Dropout(rate = taxa_dropout)
        if qtd_classes == 2:
            self.last_dense    =    layers.Dense(units    =    1,
activation = 'sigmoid')
        else:
            #problemas de classificação com mais de 2 classes
utilizar a softmax
            self.last_dense     =     layers.Dense(units     =
qtd_classes,          activation          =          'softmax',
kernel_regularizer=keras.regularizers.l2(0.01))

    def call(self, inputs, training):
        x = inputs
        x_1 = self.bigram(x)
        x_1 = self.pool(x_1)
        x_2 = self.trigram(x)
        x_2 = self.pool(x_2)
        x_3 = self.fourgram(x)
        x_3 = self.pool(x_3)
        x_4 = self.fivegram(x)
        x_4 = self.pool(x_4)

        merged = tf.concat([x_1, x_2, x_3,x_4], axis = -1) #
(batch_size, 3*qtd_filtros)
        print(merged.shape)
        merged = self.dense_1(merged)
        merged = self.dropout(merged, training)
        merged = self.dense_2(merged)
```

```
        merged = self.dropout(merged, training)
        merged = self.dense_3(merged)
        merged = self.dropout(merged, training)
        output = self.last_dense(merged)

        return output
```

Para classificar os textos dos itens do Enem inicialmente foi utilizada uma Rede Neural Convolucional treinada com a seguinte arquitetura:

```
Model: "dcnn"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv1d_4 (Conv1D)            multiple                  6432

 conv1d_5 (Conv1D)            multiple                  9632

 conv1d_6 (Conv1D)            multiple                  12832

 conv1d_7 (Conv1D)            multiple                  16032

 global_max_pooling1d_1 (Glo  multiple                  0
 balMaxPooling1D)

 dense_4 (Dense)              multiple                  8256

 dense_5 (Dense)              multiple                  4160

 dense_6 (Dense)              multiple                  4160

 dropout_1 (Dropout)          multiple                  0

 dense_7 (Dense)              multiple                  260

=================================================================
Total params: 61,764
Trainable params: 61,764
Non-trainable params: 0
_________________________
```

A rede neural e possui as seguintes camadas:

- Uma camada Conv1D chamada "conv1d_4" com saída múltipla (ou seja, várias saídas) e 6.432 parâmetros.
- Uma camada Conv1D chamada "conv1d_5" com saída múltipla e 9.632 parâmetros.
- Uma camada Conv1D chamada "conv1d_6" com saída múltipla e 12.832 parâmetros.
- Uma camada Conv1D chamada "conv1d_7" com saída múltipla e 16.032 parâmetros.
- Uma camada GlobalMaxPooling1D que faz o pooling máximo global dos valores de entrada. Essa camada tem saída múltipla e nenhum parâmetro.
- Três camadas Dense (totalmente conectadas) chamadas "dense_4", "dense_5" e "dense_6", todas com saída múltipla e respectivamente 8.256, 4.160 e 4.160 parâmetros.
- Uma camada Dropout que aplica a técnica de regularização Dropout para evitar overfitting. Essa camada não possui saída e nenhum parâmetro.
- Uma camada Dense chamada "dense_7" com saída múltipla e 260 parâmetros.

No final, a rede possui um total de 61.764 parâmetros, que são todos treináveis, e nenhuma camada não treinável.

Utilizou-se regularização L2 na primeira camada convolucional e na última camada densa. A regularização L2 é uma técnica usada para evitar o overfitting, ou seja, o modelo aprender demais o conjunto de treinamento, e não generalizar bem para novos dados. A regularização L2 adiciona um termo à função de custo da rede neural, que penaliza pesos grandes. Especificamente, a função de custo adiciona a soma dos quadrados dos pesos multiplicada por um parâmetro de regularização lambda. Essa penalização faz com que a rede favoreça pesos menores, o que pode ajudar a evitar o overfitting.

Como número de classes foi diferente de 2, a rede neural foi usada para classificação multiclasse e a função de perda utilizada foi a sparse_categorical_crossentropy como o otimizador Adam. Além disso, o desempenho será avaliado usando a métrica de acurácia.

Primeiramente o modelo para o parâmetro a (discriminação) foi treinado com os vetores de word embedding treinados somente com os textos das questões coletadas utilizando skip-gram. Os hiperparâmetros da rede neural foram:

- Quantidade de filtros convolucionais: 32;
- Quantidade de neurônios da camada densa: 64;
- Tamanho do batch: 100;
- Taxa de dropout: 0.2;
- Quantidade de épocas: 200;

A chamada para o método “fit” foi a seguinte:

```
#treinar a rede neural convolucional
history = Dcnn.fit(X_tensor, y, batch_size = tamanho_batch,
epochs = qtd_epocas, verbose = 1, validation_split = 0.2,
shuffle = True)
```

A vantagem de usar o parâmetro shuffle no método fit é que ele embaralha os dados de treinamento a cada época, o que pode melhorar a generalização do modelo e prevenir overfitting. Isso acontece porque o modelo não é treinado com um conjunto de dados sempre na mesma ordem, o que pode evitar que ele memorize as informações e não generalize corretamente.

Para uma configuração com os hiperparâmetros como a relatada na última época de treinamento a função de perda (loss) calculada no conjunto de treinamento função de perda foi 0.2858 e a acurácia (accuracy) foi 0.9149. Para o conjunto de validação, a função de perda foi 4.8017 e a acurácia foi 0.2595.

Para os parâmetros b e c o treinamento foi semelhante. Com o uso do Word embedding cc.pt.300 tanto a acurácia de treinamento quanto a acurácia de validação foram baixas.

Os arquivos notebook de treinamento e avaliação dos modelos são os seguintes:

Arquivo notebook	Parâmetro da TRI	Word Embedding
treinamento-modelo-param-a-we-sg.ipynb	a	skip-gram
treinamento-modelo-param-b-we-sg.ipynb	b	skip-gram
treinamento-modelo-param-c-we-sg.ipynb	c	skip-gram
treinamento-modelo-param-a-we-cc.pt.300	a	cc.pt.300
treinamento-modelo-param-b-we-cc.pt.300	b	cc.pt.300
treinamento-modelo-param-c-we-cc.pt.300	c	cc.pt.300

8. Avaliação dos Modelos de Aprendizado de Máquina e Discussão dos Resultados

Os dados de teste abrangeram 25% dos dados do conjunto de dados ou 481 pontos de dados. Obtendo somente o loss e a acurácia do modelo executando o código a seguir de avaliação inicial:

```
#avalia o modelo com o conjunto de teste
results = Dcnn.evaluate(X_test, y_test, batch_size=tamanho_batch)
print(results)
```

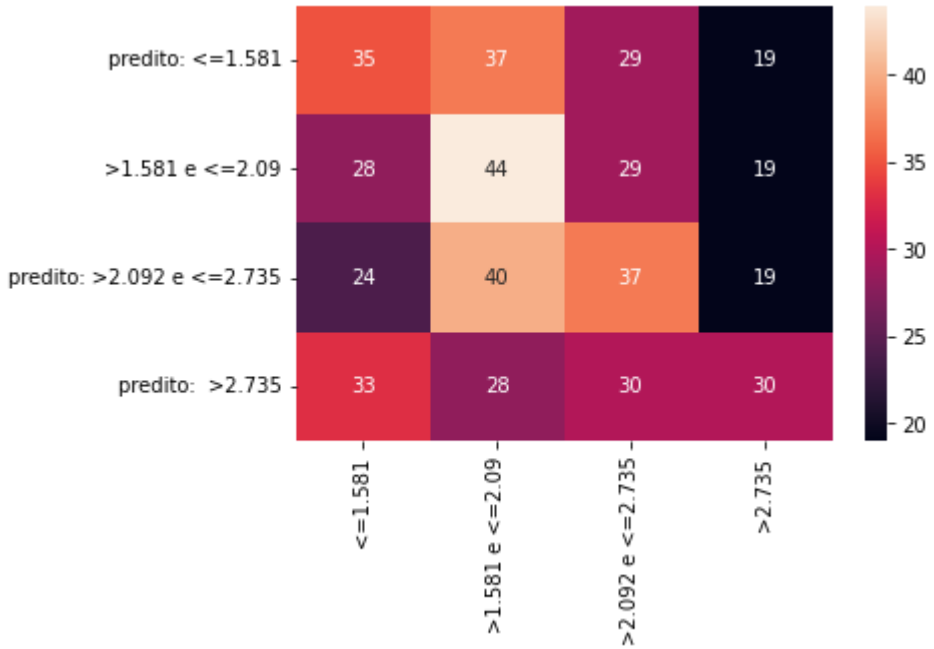
Obtiveram-se as classes preditas executando o seguinte código:

```
y_pred_test = Dcnn.predict(X_test)
#valores previstos
y_pred_test_values = []
for y_ in y_pred_test:
    #obtem o valor máximo do array de probabilidades
    y_pred_test_values.append(np.argmax(y_))

y_pred_test_values = np.array(y_pred_test_values)
y_pred_test_values
```

É apresentado um loss de 4.1610 e uma acurácia de 0.3035, coerente com a acurácia de validação no treinamento.

A matriz de confusão é a seguinte:

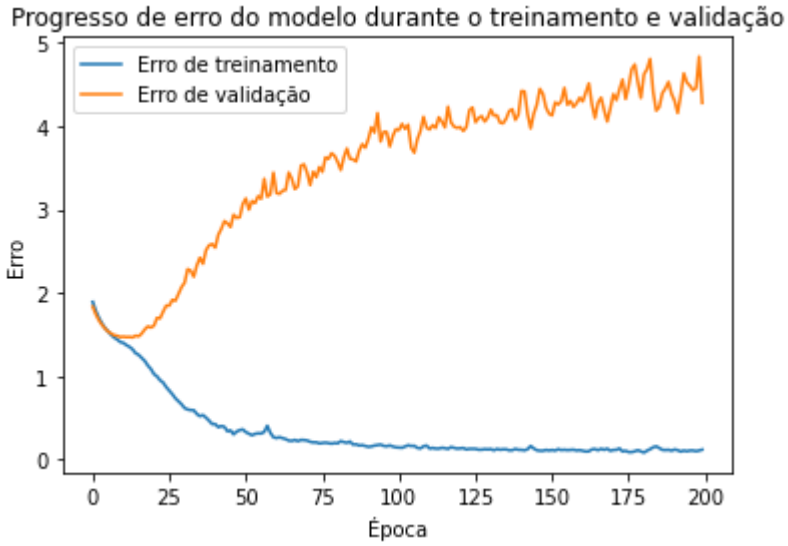


O classification report é o seguinte:

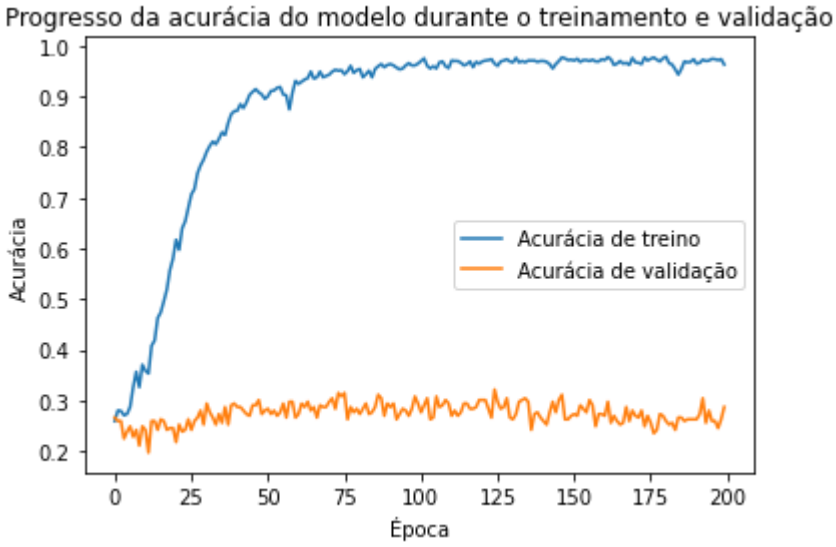
	precision	recall	f1-score	support
0	0.29	0.29	0.29	120
1	0.30	0.37	0.33	120
2	0.30	0.31	0.30	120
3	0.34	0.25	0.29	121
accuracy			0.30	481
macro avg	0.31	0.30	0.30	481
weighted avg	0.31	0.30	0.30	481

Com base nas métricas fornecidas, pode-se que o modelo tem baixa precisão, baixo recall e baixo f1-score, o que indica que ele está tendo dificuldade em classificar corretamente os exemplos positivos e ainda não está conseguindo identificar a maioria deles, sendo que ele foi testado em 120/1 exemplos da classe.

O gráfico a seguir pode ser utilizado para comparar o erro de treinamento e o erro de validação. É possível verificar que ocorreu Overfitting: o erro de treinamento está caindo enquanto o erro de validação está aumentando, é provável que o modelo esteja sofrendo de overfitting. Isso significa que o modelo está se ajustando muito bem aos dados de treinamento, mas não está generalizando bem para os dados de validação.



A análise do gráfico de comparação da acurácia de treinamento e acurácia de validação também indica Overfitting pois o gráfico de treinamento mostra uma acurácia muito alta e o gráfico de validação mostra uma acurácia muito baixa, ou seja, o modelo está memorizando o conjunto de treinamento, mas não é geral o suficiente para prever corretamente dados desconhecidos.



Utilizando o método bootstrap para estimar a incerteza (desvio padrão) em métricas obtém-se o seguinte:

```
accuracy                    : 0.30 ± 0.02
macro avg        f1-score   : 0.30 ± 0.02
macro avg        recall     : 0.30 ± 0.02
macro avg        precision  : 0.31 ± 0.02
```

```
weighted avg     f1-score  : 0.30 ± 0.02
weighted avg     recall    : 0.30 ± 0.02
weighted avg     precision : 0.31 ± 0.02
```

O método de bootstrap é uma técnica para estimar a incerteza em métricas com base em amostragem com reposição. A idéia é que, ao realizarmos várias amostragens de nosso conjunto de dados com reposição, podemos obter diferentes valores para nossa métrica. Então, podemos estimar a variabilidade da nossa métrica e, portanto, seu desvio padrão.

O modelo foi treinado variando os hiperparâmetros como quantidade de filtros, camadas, épocas, batchs etc, porém, o desempenho continuava o mesmo, o que sugere que o conjunto de dados disponíveis não é suficientemente grande e representativo, é preciso obter mais dados de treinamento e isso não é viável no momento para o projeto em questão. Apesar de o desempenho da rede neural convolucional ter sido ruim houve a oportunidade de se passar por todos os passos de treinamento e avaliação do modelo.

Para os parâmetros b e c os resultados foram semelhantes.

9. Conclusão

Conclui-se que, para o **parâmetro "a"** (discriminação) os modelos possuem os seguintes valores para as métricas:

skip-gram:

```
accuracy                  : 0.30 ± 0.02
macro avg        f1-score  : 0.30 ± 0.02
macro avg        recall    : 0.30 ± 0.02
macro avg        precision : 0.31 ± 0.02
weighted avg     f1-score  : 0.30 ± 0.02
weighted avg     recall    : 0.30 ± 0.02
weighted avg     precision : 0.31 ± 0.02
```

Essas são métricas de avaliação de um modelo de classificação. Os valores apresentados são médias e desvios padrão (±) dessas métricas calculadas para todas as classes.

- "accuracy" é a acurácia do modelo, ou seja, a proporção de previsões corretas em relação ao total de previsões.

- "macro avg f1-score", "macro avg recall" e "macro avg precision" são a média ponderada das métricas F1-score, recall (sensibilidade) e precision (valor preditivo positivo) com ponderação feita de forma igualitária para todas as classes, independentemente do número de amostras em cada classe.
- "weighted avg f1-score", "weighted avg recall" e "weighted avg precision" são métricas semelhantes às anteriores, mas com a ponderação feita de acordo com a proporção de amostras em cada classe. Essas métricas são mais indicadas quando as classes não estão balanceadas, ou seja, quando há muito mais amostras em algumas classes do que em outras.
- O valor ± 0.02 indica o desvio padrão das métricas em relação à média calculada. Quanto menor o desvio padrão, mais consistente é o desempenho do modelo em relação a cada classe.

cc.pt.300:

```
accuracy                  : 0.24 ± 0.02
macro avg        f1-score  : 0.22 ± 0.02
macro avg        recall    : 0.24 ± 0.02
macro avg        precision : 0.27 ± 0.03
weighted avg     f1-score  : 0.21 ± 0.02
weighted avg     recall    : 0.24 ± 0.02
weighted avg     precision : 0.27 ± 0.03
```

Conclui-se que, para o **parâmetro "b"** (facilidade) os modelos possuem os seguintes valores para as métricas:

skip-gram:

```
accuracy                  : 0.23 ± 0.02
macro avg        f1-score  : 0.23 ± 0.02
macro avg        recall    : 0.24 ± 0.02
macro avg        precision : 0.23 ± 0.02
weighted avg     f1-score  : 0.22 ± 0.02
weighted avg     recall    : 0.23 ± 0.02
weighted avg     precision : 0.22 ± 0.02
```

cc.pt.300:

```
accuracy                  : 0.27 ± 0.02
macro avg        f1-score  : 0.25 ± 0.02
macro avg        recall    : 0.28 ± 0.02
macro avg        precision : 0.28 ± 0.02
weighted avg     f1-score  : 0.25 ± 0.02
weighted avg     recall    : 0.27 ± 0.02
```

```
weighted avg     precision : 0.28 ± 0.03
```

Conclui-se que, para o **parâmetro "c"** (acerto ao acaso) os modelos possuem os seguintes valores para as métricas:

skip-gram:

```
accuracy                  : 0.25 ± 0.02
macro avg        f1-score  : 0.25 ± 0.02
macro avg        recall    : 0.25 ± 0.02
macro avg        precision : 0.25 ± 0.02
weighted avg     f1-score  : 0.25 ± 0.02
weighted avg     recall    : 0.25 ± 0.02
weighted avg     precision : 0.25 ± 0.02
```

cc.pt.300:

```
accuracy                  : 0.26 ± 0.02
macro avg        f1-score  : 0.23 ± 0.02
macro avg        recall    : 0.26 ± 0.02
macro avg        precision : 0.26 ± 0.03
weighted avg     f1-score  : 0.23 ± 0.02
weighted avg     recall    : 0.26 ± 0.02
weighted avg     precision : 0.26 ± 0.03
```

Conclui-se que houve overffiting no treinamento e o conjunto de dados disponíveis não é suficientemente grande e representativo, é preciso obter mais dados de treinamento e isso não é viável no momento para o projeto em questão. Mas, apesar de o desempenho da rede neural convolucional ter sido ruim houve a oportunidade de se passar por todos os passos de treinamento e avaliação de um modelo.