[HG logo](https://eaglemind.ai/ai/) [Chat Logs](https://eaglemind.ai/ai/chat) [Templates](https://eaglemind.ai/ai/templates/) [Logout](https://eaglemind.ai/ai/accounts/logout/)

Template: EagleMind\_Alpha

Haut du formulaire

Private:  Label:  Preference:

Bas du formulaire

Haut du formulaire

Bas du formulaire

▼

* [UNLABELLED](https://eaglemind.ai/ai/chat/1299)
* [UNLABELLED](https://eaglemind.ai/ai/chat/1300)
* [UNLABELLED](https://eaglemind.ai/ai/chat/1301)

hello

**EagleMind**

EagleMind> 1: Hello! How can I assist you today?

1) i have this date:  
  
<div>  
<style scoped>  
.dataframe tbody tr th:only-of-type {  
vertical-align: middle;  
}  
  
.dataframe tbody tr th {  
vertical-align: top;  
}  
  
.dataframe thead th {  
text-align: right;  
}  
</style>  
<table border="1" class="dataframe">  
<thead>  
<tr style="text-align: right;">  
<th></th>  
<th>total\_count</th>  
<th>%\_total</th>  
<th>|</th>  
<th>S</th>  
<th>%\_S\_total</th>  
<th>diff\_%\_S\_total</th>  
<th>|</th>  
<th>not\_S</th>  
<th>%\_not\_S\_total</th>  
<th>|</th>  
<th>%\_S\_label</th>  
<th>%\_not\_S\_label</th>  
<th>original\_label</th>  
</tr>  
</thead>  
<tbody>  
<tr>  
<th>0</th>  
<td>8285</td>  
<td>29.86</td>  
<td>|</td>  
<td>4559</td>  
<td>35.64</td>  
<td>5.78</td>  
<td>|</td>  
<td>3726</td>  
<td>24.91</td>  
<td>|</td>  
<td>55.0</td>  
<td>45.0</td>  
<td>21</td>  
</tr>  
<tr>  
<th>1</th>  
<td>948</td>  
<td>3.42</td>  
<td>|</td>  
<td>805</td>  
<td>6.29</td>  
<td>2.87</td>  
<td>|</td>  
<td>143</td>  
<td>0.96</td>  
<td>|</td>  
<td>85.0</td>  
<td>15.0</td>  
<td>25</td>  
</tr>  
<tr>  
<th>2</th>  
<td>2273</td>  
<td>8.19</td>  
<td>|</td>  
<td>1382</td>  
<td>10.80</td>  
<td>2.61</td>  
<td>|</td>  
<td>891</td>  
<td>5.96</td>  
<td>|</td>  
<td>61.0</td>  
<td>39.0</td>  
<td>2</td>  
</tr>  
<tr>  
<th>3</th>  
<td>1715</td>  
<td>6.18</td>  
<td>|</td>  
<td>1113</td>  
<td>8.70</td>  
<td>2.52</td>  
<td>|</td>  
<td>602</td>  
<td>4.03</td>  
<td>|</td>  
<td>65.0</td>  
<td>35.0</td>  
<td>18</td>  
</tr>  
<tr>  
<th>4</th>  
<td>703</td>  
<td>2.53</td>  
<td>|</td>  
<td>540</td>  
<td>4.22</td>  
<td>1.69</td>  
<td>|</td>  
<td>163</td>  
<td>1.09</td>  
<td>|</td>  
<td>77.0</td>  
<td>23.0</td>  
<td>1</td>  
</tr>  
<tr>  
<th>5</th>  
<td>994</td>  
<td>3.58</td>  
<td>|</td>  
<td>652</td>  
<td>5.10</td>  
<td>1.52</td>  
<td>|</td>  
<td>342</td>  
<td>2.29</td>  
<td>|</td>  
<td>66.0</td>  
<td>34.0</td>  
<td>6</td>  
</tr>  
<tr>  
<th>6</th>  
<td>467</td>  
<td>1.68</td>  
<td>|</td>  
<td>391</td>  
<td>3.06</td>  
<td>1.38</td>  
<td>|</td>  
<td>76</td>  
<td>0.51</td>  
<td>|</td>  
<td>84.0</td>  
<td>16.0</td>  
<td>24</td>  
</tr>  
<tr>  
<th>7</th>  
<td>386</td>  
<td>1.39</td>  
<td>|</td>  
<td>324</td>  
<td>2.53</td>  
<td>1.14</td>  
<td>|</td>  
<td>62</td>  
<td>0.41</td>  
<td>|</td>  
<td>84.0</td>  
<td>16.0</td>  
<td>5</td>  
</tr>  
<tr>  
<th>8</th>  
<td>2918</td>  
<td>10.52</td>  
<td>|</td>  
<td>1462</td>  
<td>11.43</td>  
<td>0.91</td>  
<td>|</td>  
<td>1456</td>  
<td>9.74</td>  
<td>|</td>  
<td>50.0</td>  
<td>50.0</td>  
<td>19</td>  
</tr>  
<tr>  
<th>9</th>  
<td>499</td>  
<td>1.80</td>  
<td>|</td>  
<td>347</td>  
<td>2.71</td>  
<td>0.91</td>  
<td>|</td>  
<td>152</td>  
<td>1.02</td>  
<td>|</td>  
<td>70.0</td>  
<td>30.0</td>  
<td>9</td>  
</tr>  
<tr>  
<th>10</th>  
<td>1402</td>  
<td>5.05</td>  
<td>|</td>  
<td>717</td>  
<td>5.60</td>  
<td>0.55</td>  
<td>|</td>  
<td>685</td>  
<td>4.58</td>  
<td>|</td>  
<td>51.0</td>  
<td>49.0</td>  
<td>11</td>  
</tr>  
<tr>  
<th>11</th>  
<td>206</td>  
<td>0.74</td>  
<td>|</td>  
<td>160</td>  
<td>1.25</td>  
<td>0.51</td>  
<td>|</td>  
<td>46</td>  
<td>0.31</td>  
<td>|</td>  
<td>78.0</td>  
<td>22.0</td>  
<td>4</td>  
</tr>  
<tr>  
<th>12</th>  
<td>387</td>  
<td>1.39</td>  
<td>|</td>  
<td>237</td>  
<td>1.85</td>  
<td>0.46</td>  
<td>|</td>  
<td>150</td>  
<td>1.00</td>  
<td>|</td>  
<td>61.0</td>  
<td>39.0</td>  
<td>0</td>  
</tr>  
<tr>  
<th>13</th>  
<td>178</td>  
<td>0.64</td>  
<td>|</td>  
<td>132</td>  
<td>1.03</td>  
<td>0.39</td>  
<td>|</td>  
<td>46</td>  
<td>0.31</td>  
<td>|</td>  
<td>74.0</td>  
<td>26.0</td>  
<td>16</td>  
</tr>  
<tr>  
<th>14</th>  
<td>184</td>  
<td>0.66</td>  
<td>|</td>  
<td>128</td>  
<td>1.00</td>  
<td>0.34</td>  
<td>|</td>  
<td>56</td>  
<td>0.37</td>  
<td>|</td>  
<td>70.0</td>  
<td>30.0</td>  
<td>3</td>  
</tr>  
<tr>  
<th>15</th>  
<td>107</td>  
<td>0.39</td>  
<td>|</td>  
<td>92</td>  
<td>0.72</td>  
<td>0.33</td>  
<td>|</td>  
<td>15</td>  
<td>0.10</td>  
<td>|</td>  
<td>86.0</td>  
<td>14.0</td>  
<td>8</td>  
</tr>  
<tr>  
<th>16</th>  
<td>92</td>  
<td>0.33</td>  
<td>|</td>  
<td>83</td>  
<td>0.65</td>  
<td>0.32</td>  
<td>|</td>  
<td>9</td>  
<td>0.06</td>  
<td>|</td>  
<td>90.0</td>  
<td>10.0</td>  
<td>23</td>  
</tr>  
<tr>  
<th>17</th>  
<td>542</td>  
<td>1.95</td>  
<td>|</td>  
<td>286</td>  
<td>2.24</td>  
<td>0.29</td>  
<td>|</td>  
<td>256</td>  
<td>1.71</td>  
<td>|</td>  
<td>53.0</td>  
<td>47.0</td>  
<td>14</td>  
</tr>  
<tr>  
<th>18</th>  
<td>504</td>  
<td>1.82</td>  
<td>|</td>  
<td>258</td>  
<td>2.02</td>  
<td>0.20</td>  
<td>|</td>  
<td>246</td>  
<td>1.64</td>  
<td>|</td>  
<td>51.0</td>  
<td>49.0</td>  
<td>20</td>  
</tr>  
<tr>  
<th>19</th>  
<td>97</td>  
<td>0.35</td>  
<td>|</td>  
<td>55</td>  
<td>0.43</td>  
<td>0.08</td>  
<td>|</td>  
<td>42</td>  
<td>0.28</td>  
<td>|</td>  
<td>57.0</td>  
<td>43.0</td>  
<td>17</td>  
</tr>  
<tr>  
<th>20</th>  
<td>93</td>  
<td>0.34</td>  
<td>|</td>  
<td>14</td>  
<td>0.11</td>  
<td>-0.23</td>  
<td>|</td>  
<td>79</td>  
<td>0.53</td>  
<td>|</td>  
<td>15.0</td>  
<td>85.0</td>  
<td>10</td>  
</tr>  
<tr>  
<th>21</th>  
<td>115</td>  
<td>0.41</td>  
<td>|</td>  
<td>21</td>  
<td>0.16</td>  
<td>-0.25</td>  
<td>|</td>  
<td>94</td>  
<td>0.63</td>  
<td>|</td>  
<td>18.0</td>  
<td>82.0</td>  
<td>27</td>  
</tr>  
<tr>  
<th>22</th>  
<td>136</td>  
<td>0.49</td>  
<td>|</td>  
<td>21</td>  
<td>0.16</td>  
<td>-0.33</td>  
<td>|</td>  
<td>115</td>  
<td>0.77</td>  
<td>|</td>  
<td>15.0</td>  
<td>85.0</td>  
<td>15</td>  
</tr>  
<tr>  
<th>23</th>  
<td>1121</td>  
<td>4.04</td>  
<td>|</td>  
<td>441</td>  
<td>3.45</td>  
<td>-0.59</td>  
<td>|</td>  
<td>680</td>  
<td>4.55</td>  
<td>|</td>  
<td>39.0</td>  
<td>61.0</td>  
<td>26</td>  
</tr>  
<tr>  
<th>24</th>  
<td>262</td>  
<td>0.94</td>  
<td>|</td>  
<td>22</td>  
<td>0.17</td>  
<td>-0.77</td>  
<td>|</td>  
<td>240</td>  
<td>1.60</td>  
<td>|</td>  
<td>8.0</td>  
<td>92.0</td>  
<td>7</td>  
</tr>  
<tr>  
<th>25</th>  
<td>1295</td>  
<td>4.67</td>  
<td>|</td>  
<td>494</td>  
<td>3.86</td>  
<td>-0.81</td>  
<td>|</td>  
<td>801</td>  
<td>5.36</td>  
<td>|</td>  
<td>38.0</td>  
<td>62.0</td>  
<td>22</td>  
</tr>  
<tr>  
<th>26</th>  
<td>524</td>  
<td>1.89</td>  
<td>|</td>  
<td>93</td>  
<td>0.73</td>  
<td>-1.16</td>  
<td>|</td>  
<td>431</td>  
<td>2.88</td>  
<td>|</td>  
<td>18.0</td>  
<td>82.0</td>  
<td>12</td>  
</tr>  
<tr>  
<th>27</th>  
<td>1316</td>  
<td>4.74</td>  
<td>|</td>  
<td>127</td>  
<td>0.99</td>  
<td>-3.75</td>  
<td>|</td>  
<td>1189</td>  
<td>7.95</td>  
<td>|</td>  
<td>10.0</td>  
<td>90.0</td>  
<td>13</td>  
</tr>  
</tbody>  
</table>  
</div>  
  
2)i have thius challenge  
  
I have a data challenge where i must classify by debiasing the dataset. here is the starting kit :  
  
  
JUPYTER  
FAQ  
Data Challenge : Introduction to Machine Learning (BGDIA703)  
Authors:  
Charlotte Laclau (charlotte.laclau@telecom-paris.fr)  
This year's challenge is about text classification. For privacy reasons, you are only provided with the embedding learned on the original documents.  
  
Fair document classification  
The task is straightforward: assign the correct category to a text. This is thus a multi-class classification task with 28 classes to choose from.  
  
The most adopted paradigm consists in training a network f:X→Rd  
which, from a given document x∈X  
, extracts a feature vector z∈Rd  
which synthetizes the relevant caracteristics of doc  
. The diagnostic phase then consists, from an document x  
, to predict the label of the document based on the extracted features z  
. In this data challenge d=768  
.  
  
We directly provide you the embedding of each text (learned with BERT).  
  
The goal of this competition is to design a solution that is both accurate for predicting the label as well as fair with respect to some sensitive attribute (e.g. gender). Fairness in this context means that the model should not be biased toward a certain minority group present in the data. We explain this paradigm further in the evaluation part.  
  
Downloading the Data:  
You can download the Dataset and evaluator script from the below links: https://partage.imt.fr/index.php/s/3M3Mn3sN3TNSDEp  
  
https://partage.imt.fr/index.php/s/CfajSjkAiq2oCLF  
  
After unzipping the file, you have one pickle file named data-challenge-student.pickle. In this file you have one dictionnary that contains the training set X\_train, the test set X\_test, the training label Y\_train, and the information about the sensitive attribute in S\_train and S\_test.  
  
import pickle  
import pandas as pd  
with open('data-challenge-student.pickle', 'rb') as handle:  
# dat = pickle.load(handle)  
dat = pd.read\_pickle(handle)  
X = dat['X\_train']  
Y = dat['Y']  
S = dat['S\_train']  
Evaluation  
First of all, the accuracy of the solutions are evaluated according to the Macro F1 metric, The Macro F1 score is simply the arithmetic average of the F1 score for each class.  
  
We will also analyse proposed solutions according to their fairness with respect to the provided sensitive attribute (S  
). In other words, we want you to design a solution that is not biased towards one group in particular. To be specific, we will use (1-equal opportunity gap) between protected groups. A fair model is a model where this criteria is close to 1.  
  
Overall, your model should satisfy both criteria so the evaluation metric is the average between the macro F1 and the fairness criteria.  
  
The file evaluator.py contains the required functions to compute the final score on which you will be ranked.  
  
Baseline  
Let us use a logistic regression as our naive baseline model. Note that this model does not take into accout the sensitive attribute S  
. It will only be used for the evaluation.  
  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
from evaluator import \*  
# Train the logistic regression  
X\_train, X\_test, Y\_train, Y\_test, S\_train, S\_test = train\_test\_split(X, Y, S, test\_size=0.3, random\_state=42)  
clf = LogisticRegression(random\_state=0, max\_iter=5000).fit(X\_train, Y\_train)  
Y\_pred = clf.predict(X\_test)  
eval\_scores, confusion\_matrices\_eval = gap\_eval\_scores(Y\_pred, Y\_test, S\_test, metrics=['TPR'])  
final\_score = (eval\_scores['macro\_fscore']+ (1-eval\_scores['TPR\_GAP']))/2  
print(final\_score)  
0.7289043899708004  
Preparing the submission file  
Now we are ready to prepare a submission file. In the pickle you have access to some additional test data (X\_test, S\_test) and you should submit your prediction for Y. Note that with the current model, you do not need S\_test but we provide it to you in case you want to use it in a debiasing strategy.  
  
# Load the "true" test data  
X\_test = dat['X\_test']  
S\_test = dat['S\_test']  
# Classify the provided test data with you classifier  
y\_test = clf.predict(X\_test)  
results=pd.DataFrame(y\_test, columns= ['score'])  
  
results.to\_csv("Data\_Challenge\_MDI\_341.csv", header = None, index = None)  
# np.savetxt('y\_test\_challenge\_student.txt', y\_test, delimiter=',')  
# Good luck !  
This website does not host notebooks, it only renders notebooks available on other websites.  
  
Delivered by Fastly, Rendered by OVHcloud  
  
nbviewer GitHub repository.  
  
nbviewer version: 8b013f7  
  
nbconvert version: 7.2.3  
  
Rendered a few seconds ago  
  
3) can you analyze the data accordingly

**EagleMind**

EagleMind> 2: To analyze the data and address the challenge you've presented, let's break down the tasks and approach each systematically.  
  
### Analyzing the Data  
  
The data you've provided is in a tabular format with several key columns:  
  
- \*\*total\_count\*\*: The total number of documents.  
- \*\*%\_total\*\*: The percentage of the total documents.  
- \*\*S\*\*: The count of documents with a sensitive attribute.  
- \*\*%\_S\_total\*\*: The percentage of documents with the sensitive attribute out of the total documents.  
- \*\*diff\_%\_S\_total\*\*: The difference in percentage points between documents with and without the sensitive attribute.  
- \*\*not\_S\*\*: The count of documents without the sensitive attribute.  
- \*\*%\_not\_S\_total\*\*: The percentage of documents without the sensitive attribute out of the total documents.  
- \*\*%\_S\_label\*\* and \*\*%\_not\_S\_label\*\*: The percentage distribution of labels for documents with and without the sensitive attribute.  
- \*\*original\_label\*\*: The original label of the documents.  
  
#### Observations:  
- The data seems to be already categorized based on a sensitive attribute (S), with a clear division between documents with and without this attribute.  
- There's a notable difference in the distribution of documents with and without the sensitive attribute across different labels, which could indicate potential bias in the dataset.  
  
### Addressing the Challenge  
  
The challenge involves creating a model that can classify documents into one of 28 categories while also ensuring fairness with respect to the sensitive attribute. The fairness criterion is measured by the (1-equal opportunity gap) between protected groups, aiming for a model where this criterion is close to 1.  
  
#### Steps to Address the Challenge:  
  
1. \*\*Data Preprocessing\*\*:  
- Ensure the data is clean and ready for modeling. This might involve handling missing values, normalizing data, or encoding categorical variables if necessary.  
  
2. \*\*Feature Engineering\*\*:  
- Consider creating new features from the existing data that could help improve model performance. For example, differences in document length or the use of specific keywords might be relevant.  
  
3. \*\*Model Selection\*\*:  
- While the baseline model is a logistic regression, exploring other models like Random Forest, Gradient Boosting Machines, or even neural networks could yield better performance.  
  
4. \*\*Debiasing Strategies\*\*:  
- Implement strategies to mitigate bias with respect to the sensitive attribute. This could involve techniques like re-weighting training examples, using adversarial training to learn fair representations, or post-processing model predictions to ensure fairness.  
  
5. \*\*Evaluation\*\*:  
- Use the provided `evaluator.py` script to assess both the accuracy (Macro F1 score) and fairness (1-equal opportunity gap) of your model. Aim for a balance between these two metrics to maximize the final score.  
  
6. \*\*Iteration\*\*:  
- It's likely that the first model won't be perfect. Use insights from your evaluations to iteratively improve the model. This could involve tuning hyperparameters, trying different debiasing strategies, or even revisiting your feature engineering process.  
  
7. \*\*Final Prediction and Submission\*\*:  
- Once satisfied with your model's performance and fairness, use it to predict the labels for the provided test data (`X\_test`). Submit these predictions as instructed.  
  
### Additional Tips:  
- \*\*Exploratory Data Analysis (EDA)\*\*: Before diving into modeling, perform EDA to understand the data better. Look for patterns, anomalies, or any insights that could inform your modeling strategy.  
- \*\*Model Explainability\*\*: Utilizing tools or techniques to interpret your model's predictions can provide insights into how it's making decisions and whether any features are disproportionately influencing those decisions.  
  
This challenge combines the technical aspects of machine learning with the ethical considerations of fairness and bias, making it a valuable learning experience. Good luck!

**EagleMind**

EagleMind> 2: To analyze the data and address the challenge as described, we'll break down the process into structured steps according to the Operational Process Framework (OPF) and integrate insights from the Value-Driven Adoption Model (VDAM) and the Query Causal Model (QCM). This approach ensures a comprehensive analysis that considers both the technical and ethical aspects of the challenge.  
  
### Step 1: Understanding the Data Structure  
The provided HTML table structure indicates a dataset with multiple attributes, including total counts, percentages, and labels, alongside sensitive attributes (S/not\_S) indicating a group membership which could be related to the fairness aspect of the challenge.  
  
### Step 2: Data Challenge Objective  
- \*\*Primary Objective:\*\* Classify texts into one of 28 categories based on their embeddings, ensuring high accuracy (Macro F1 score).  
- \*\*Secondary Objective:\*\* Ensure fairness in classification, minimizing bias towards any group represented by the sensitive attribute (S).  
  
### Step 3: Preparing the Data  
1. \*\*Load the Dataset:\*\* Utilize the provided pickle file to load the training and test sets, labels, and sensitive attributes.  
2. \*\*Data Exploration:\*\* Examine the distribution of classes and the representation of the sensitive attribute across the dataset to identify any initial biases or imbalances.  
  
### Step 4: Developing a Baseline Model  
- Implement a logistic regression model as a baseline, noting its performance in terms of accuracy and fairness.  
- This model does not consider the sensitive attribute for prediction but will be evaluated for fairness based on the sensitive attribute's impact on predictions.  
  
### Step 5: Enhancing Fairness and Accuracy  
1. \*\*Debiasing Strategies:\*\* Explore techniques such as re-weighting training examples, modifying the learning objective to include fairness constraints, or post-processing predictions to adjust for fairness.  
2. \*\*Model Selection:\*\* Consider alternative models or ensemble methods that might offer better performance or fairness, such as decision trees, support vector machines, or neural networks.  
3. \*\*Feature Engineering:\*\* Investigate if additional features derived from the embeddings or external data sources could improve model performance or fairness.  
  
### Step 6: Evaluation and Iteration  
- Utilize the provided `evaluator.py` script to assess both the Macro F1 score and fairness criteria.  
- Iteratively refine the model by adjusting debiasing strategies, model parameters, or feature selection based on evaluation results.  
  
### Step 7: Final Model Selection  
- Choose the model that best balances accuracy and fairness according to the competition's evaluation metric (average of Macro F1 and fairness criteria).  
  
### Step 8: Submission Preparation  
- Apply the final model to the provided test data.  
- Prepare and submit the prediction results as instructed, ensuring compliance with the submission format.  
  
### Conclusion  
This structured approach, informed by OPF, VDAM, and QCM, emphasizes not only achieving high classification accuracy but also ensuring fairness across sensitive groups. By iteratively refining the model and debiasing strategies based on comprehensive evaluation criteria, you can develop a solution that meets the challenge's dual objectives of accuracy and fairness.

can you solve it?

develop point 5, what type of models can be used to do this ?

**EagleMind**

EagleMind> 3: As an AI developed by OpenAI, I'm unable to directly execute code, access external databases, or perform real-time data analysis and modeling tasks, including downloading files or running scripts from external links. However, I can guide you through the process of solving the challenge based on the information you've provided, focusing on key steps and strategies that you can implement.  
  
### Step 1: Data Preparation  
- \*\*Load the Dataset\*\*: Use the provided Python code snippet to load your dataset from the pickle file.  
**python**

import pickle

import pandas as pd

with open('data-challenge-student.pickle', 'rb') as handle:

dat = pd.read\_pickle(handle)

X\_train = dat['X\_train']

Y\_train = dat['Y']

S\_train = dat['S\_train']

### Step 2: Exploratory Data Analysis (EDA)  
- Perform EDA to understand the distribution of classes and the representation of the sensitive attribute. This can help identify any imbalances or biases in the dataset.  
- Visualize the embeddings if possible, using techniques like PCA or t-SNE, to see if there are any noticeable patterns or clusters.  
  
### Step 3: Baseline Model  
- Implement the logistic regression model as described. This serves as your baseline for performance and fairness.  
**python**

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

# Splitting the dataset for training and testing

X\_train, X\_test, Y\_train, Y\_test, S\_train, S\_test = train\_test\_split(X\_train, Y\_train, S\_train, test\_size=0.3, random\_state=42)

# Training the logistic regression model

clf = LogisticRegression(random\_state=0, max\_iter=5000).fit(X\_train, Y\_train)

### Step 4: Fairness and Accuracy Improvement  
- \*\*Model Selection\*\*: Explore different models and select the one that offers the best trade-off between accuracy and fairness. Consider models known for their performance on text data, such as SVM or neural networks.  
- \*\*Debiasing Techniques\*\*: Apply debiasing techniques such as adjusting class weights, using fairness-aware algorithms, or applying post-processing fairness corrections.  
  
### Step 5: Evaluation  
- Use the `evaluator.py` script functions to evaluate your model's performance and fairness. Aim to improve both the Macro F1 score and the fairness metric.  
**python**

Y\_pred = clf.predict(X\_test)

# Assuming eval\_scores function is defined in evaluator.py and properly imported

eval\_scores, confusion\_matrices\_eval = gap\_eval\_scores(Y\_pred, Y\_test, S\_test, metrics=['TPR'])

final\_score = (eval\_scores['macro\_fscore'] + (1 - eval\_scores['TPR\_GAP'])) / 2

print(final\_score)

### Step 6: Final Adjustments and Submission  
- Based on the evaluation, make any final adjustments to your model or its parameters to improve both accuracy and fairness.  
- Use your final model to predict labels for the provided test data and prepare your submission according to the guidelines.  
  
**python**

# Predicting on the test set

X\_test = dat['X\_test']

y\_test\_pred = clf.predict(X\_test)

# Preparing the submission file

results = pd.DataFrame(y\_test\_pred, columns=['score'])

results.to\_csv("Data\_Challenge\_MDI\_341.csv", header=None, index=None)

### Conclusion  
While I can't execute these steps for you, following this structured approach should help you develop a solution that meets the challenge's requirements for both high accuracy and fairness. Remember, iterative testing and refinement based on evaluation metrics are key to improving your model's performance.

**EagleMind**

EagleMind> 3: Developing point 5 further, let's explore various types of models that can be used to address the challenge of text classification while also considering fairness with respect to a sensitive attribute. The goal is to find a model that not only performs well in terms of accuracy (measured by the Macro F1 score) but also ensures fairness, minimizing bias towards any group represented by the sensitive attribute.  
  
### 1. Logistic Regression  
- \*\*Description:\*\* A simple yet effective baseline model for classification tasks. It's particularly useful for understanding the impact of each feature on the prediction due to its interpretability.  
- \*\*Fairness Consideration:\*\* While logistic regression itself does not include mechanisms for ensuring fairness, post-processing techniques or incorporating fairness constraints into the loss function can help mitigate bias.  
  
### 2. Decision Trees and Random Forests  
- \*\*Description:\*\* Decision trees are simple, interpretable models that split data based on certain criteria. Random forests are ensembles of decision trees that improve prediction accuracy by reducing overfitting.  
- \*\*Fairness Consideration:\*\* These models can be analyzed for fairness at each decision node, and fairness constraints can be incorporated during the tree construction process.  
  
### 3. Gradient Boosting Machines (GBM)  
- \*\*Description:\*\* An ensemble technique that builds trees in a sequential manner, where each new tree helps to correct errors made by previously trained trees. Models like XGBoost, LightGBM, and CatBoost fall into this category.  
- \*\*Fairness Consideration:\*\* Similar to decision trees, fairness can be considered by controlling the training process, though the complexity of the model may make it harder to directly interpret fairness impacts.  
  
### 4. Support Vector Machines (SVM)  
- \*\*Description:\*\* SVMs are powerful classifiers that work by finding the hyperplane that best separates different classes in the feature space. They are effective in high-dimensional spaces, making them suitable for text classification.  
- \*\*Fairness Consideration:\*\* Fairness can be addressed by adjusting the decision boundary to minimize bias, though this may require careful tuning and validation.  
  
### 5. Neural Networks  
- \*\*Description:\*\* Deep learning models, particularly those based on transformer architectures like BERT, have shown remarkable success in NLP tasks, including text classification. They can automatically learn complex representations of text data.  
- \*\*Fairness Consideration:\*\* Neural networks can incorporate fairness through techniques like adversarial debiasing, where a model is trained to perform the task while also minimizing its ability to predict the sensitive attribute.  
  
### 6. Fairness-aware Models  
- \*\*Description:\*\* Some models are specifically designed with fairness in mind, such as adversarial debiasing models, fairness constraints models (e.g., adding fairness as a constraint in the optimization problem), and representation learning models that aim to learn fair representations of the data.  
- \*\*Fairness Consideration:\*\* These models directly incorporate fairness into the model design and training process, aiming to produce fair outcomes by construction.  
  
### Choosing the Right Model  
- The choice of model depends on the specific requirements of the task, including the trade-off between accuracy and fairness, the computational resources available, and the need for model interpretability.  
- It's often beneficial to experiment with multiple models and fairness strategies to identify the combination that best meets the challenge's objectives.  
- Evaluation should not only consider the overall performance and fairness but also how the model performs across different groups represented in the sensitive attribute to ensure that fairness improvements are not achieved at the expense of any particular group.  
  
By considering these models and their approaches to fairness, you can develop a solution that is both accurate and fair, addressing the challenge's dual objectives.

code the adversarial neural network

**EagleMind**

EagleMind> 4: Implementing an adversarial neural network for debiasing involves creating two components: the main classifier that predicts the target labels and the adversary that tries to predict the sensitive attribute from the classifier's predictions. The goal is to train the classifier to make accurate predictions while also ensuring that these predictions do not allow the adversary to accurately predict the sensitive attribute, thus reducing bias.  
  
Below is a simplified example of how you might implement such a system using TensorFlow and Keras. This example assumes you have a dataset ready for training with features `X`, target labels `Y`, and sensitive attributes `S`. The architecture and specifics might need adjustments based on your exact dataset and requirements.  
  
### Step 1: Import Necessary Libraries  
  
**python**

import tensorflow as tf

from tensorflow.keras import layers, models, optimizers, losses

### Step 2: Define the Main Classifier Model  
  
**python**

def create\_classifier(input\_dim):

model = models.Sequential()

model.add(layers.Dense(128, activation='relu', input\_dim=input\_dim))

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(1, activation='sigmoid')) # Assuming binary classification

return model

### Step 3: Define the Adversary Model  
  
**python**

def create\_adversary(embedding\_dim):

model = models.Sequential()

model.add(layers.Dense(32, activation='relu', input\_dim=embedding\_dim))

model.add(layers.Dense(1, activation='sigmoid')) # Assuming binary sensitive attribute

return model

### Step 4: Combine the Models for Adversarial Training  
  
**python**

def create\_combined\_model(classifier, adversary, input\_dim):

# Make the adversary untrainable when training the combined model

adversary.trainable = False

model\_input = layers.Input(shape=(input\_dim,))

classifier\_output = classifier(model\_input)

adversary\_output = adversary(classifier\_output)

combined\_model = models.Model(inputs=model\_input, outputs=[classifier\_output, adversary\_output])

return combined\_model

### Step 5: Compile and Train the Models  
  
**python**

input\_dim = X\_train.shape[1] # Number of features

classifier = create\_classifier(input\_dim)

adversary = create\_adversary(embedding\_dim=64) # Adjust based on your classifier's last hidden layer size

# Compile the classifier

classifier.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the classifier initially

classifier.fit(X\_train, Y\_train, epochs=10, batch\_size=32)

# Create and compile the combined model

combined\_model = create\_combined\_model(classifier, adversary, input\_dim)

combined\_model.compile(optimizer='adam', loss=['binary\_crossentropy', 'binary\_crossentropy'], loss\_weights=[1, -0.1])

# Train the combined model

# Note: Y needs to be a list containing the true labels and the sensitive attribute labels

combined\_model.fit(X\_train, [Y\_train, S\_train], epochs=10, batch\_size=32)

### Step 6: Fine-tuning and Evaluation  
  
After training, you might need to fine-tune the models based on your evaluation metrics. This could involve adjusting the loss weights, changing the model architecture, or applying different training strategies to balance accuracy and fairness.  
  
### Note:  
- This example is highly simplified and might need significant adjustments for real-world applications.  
- The effectiveness of adversarial debiasing depends on factors like the complexity of the dataset, the representation capacity of the models, and the specific definitions of fairness you're aiming to achieve.  
- Always validate the fairness and accuracy of your model on a separate validation set to ensure that your debiasing efforts are effective.  
  
This code provides a starting point for implementing an adversarial neural network for debiasing. Experimentation and iteration will be key to developing a model that meets your specific fairness and accuracy goals.

**EagleMind**

EagleMind> 4: Designing an adversarial neural network for text classification with fairness considerations involves creating a model that learns to predict the correct labels while another part of the model tries to ensure that these predictions are fair with respect to a sensitive attribute. Here's a simplified conceptual outline for implementing such a model using TensorFlow and Keras. This example assumes you have a basic understanding of neural networks and TensorFlow.  
  
### Step 1: Import Necessary Libraries  
  
**python**

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense, Embedding, LSTM, Dropout, concatenate

from tensorflow.keras.optimizers import Adam

### Step 2: Define the Model Architecture  
  
This example uses a simple LSTM network for the primary task (text classification) and a feed-forward network for the adversarial task (ensuring fairness).  
  
**python**

def create\_adversarial\_model(input\_dim, embedding\_dim, lstm\_units, num\_classes, sensitive\_classes):

# Input layer for text data

text\_input = Input(shape=(input\_dim,), name='text\_input')

# Embedding layer

emb = Embedding(input\_dim=input\_dim, output\_dim=embedding\_dim)(text\_input)

# LSTM layer for text classification

lstm = LSTM(units=lstm\_units)(emb)

lstm\_dropout = Dropout(0.2)(lstm)

# Prediction layer for the main task

main\_pred = Dense(num\_classes, activation='softmax', name='main\_output')(lstm\_dropout)

# Adversarial network to ensure fairness

adv\_layer = Dense(64, activation='relu')(lstm\_dropout)

adv\_dropout = Dropout(0.2)(adv\_layer)

adv\_pred = Dense(sensitive\_classes, activation='softmax', name='adv\_output')(adv\_dropout)

# Define the model with text input and two outputs

model = Model(inputs=text\_input, outputs=[main\_pred, adv\_pred])

return model

### Step 3: Compile the Model with Custom Loss  
  
The adversarial model requires a custom loss function that promotes fairness. This example uses a simple approach where the loss from the adversarial network is subtracted from the main task's loss, encouraging the model to make predictions that are not easily distinguishable by the sensitive attribute.  
  
**python**

def compile\_model(model):

# Loss for the main task

main\_loss = tf.keras.losses.CategoricalCrossentropy()

# Loss for the adversarial task

adv\_loss = tf.keras.losses.CategoricalCrossentropy()

# Optimizer

optimizer = Adam(learning\_rate=0.001)

# Compile the model with both losses and a single optimizer

model.compile(optimizer=optimizer,

loss={'main\_output': main\_loss, 'adv\_output': adv\_loss},

loss\_weights={'main\_output': 1.0, 'adv\_output': -0.5}, # Negative weight for adversarial loss

metrics={'main\_output': 'accuracy', 'adv\_output': 'accuracy'})

### Step 4: Train the Model  
  
When training the model, you'll need to provide the data for both the main task and the sensitive attribute. This example assumes you have `X\_train`, `Y\_train`, and `S\_train` (sensitive attribute labels) available.  
  
**python**

model = create\_adversarial\_model(input\_dim=10000, embedding\_dim=100, lstm\_units=64, num\_classes=28, sensitive\_classes=2)

compile\_model(model)

# Assuming X\_train, Y\_train (one-hot encoded), and S\_train (one-hot encoded) are available

model.fit(X\_train, {'main\_output': Y\_train, 'adv\_output': S\_train}, epochs=10, batch\_size=32)

### Notes:  
- \*\*Adversarial Training\*\*: The key idea is to train the model in such a way that the sensitive attribute predictions (made by the adversarial part of the network) are as inaccurate as possible, indicating that the main predictions are fair.  
- \*\*Hyperparameters\*\*: The choice of hyperparameters (like the negative weight for adversarial loss) is crucial and may require experimentation to find the right balance between accuracy and fairness.  
- \*\*Model Complexity\*\*: This example uses a simple LSTM for demonstration purposes. Depending on your dataset, you might need a more complex architecture, such as a Transformer model, for better performance.  
  
This code provides a basic framework for building an adversarial neural network for fair text classification. Adjustments and optimizations will likely be necessary to achieve desirable results on your specific dataset.

Do you think that using a siamese network to create a new representation space would make sense, so as to use "classical debiasing methods on binary classification with debiasing". Provide detailed answer on pros and cons between these 2 methods (adversarial NN versus Siamese network)

**EagleMind**

EagleMind> 5: The choice between using an adversarial neural network (NN) approach versus a Siamese network for debiasing in binary classification tasks involves considering the specific characteristics, advantages, and limitations of each method. Both aim to mitigate bias but operate under different paradigms and mechanisms. Let's explore the pros and cons of each.  
  
### Adversarial Neural Network Approach  
  
\*\*Pros:\*\*  
1. \*\*Direct Bias Mitigation:\*\* Adversarial training explicitly aims to reduce bias by making the sensitive attribute difficult to predict from the model's outputs, directly addressing fairness.  
2. \*\*Flexibility:\*\* This approach can be integrated with various neural network architectures, making it adaptable to different tasks beyond binary classification.  
3. \*\*Dynamic Adjustment:\*\* The adversarial component can dynamically adjust during training to continuously counteract bias as the primary model learns.  
  
\*\*Cons:\*\*  
1. \*\*Complex Training Dynamics:\*\* Balancing the adversarial game between the main model and the adversary can be challenging, potentially leading to unstable training dynamics.  
2. \*\*Hyperparameter Sensitivity:\*\* The effectiveness of debiasing can be highly sensitive to the choice of hyperparameters, such as the loss weights for the main task and adversarial task.  
3. \*\*Potential for Reduced Accuracy:\*\* In striving for fairness, there might be scenarios where the overall accuracy of the model is compromised.  
  
### Siamese Network Approach  
  
\*\*Pros:\*\*  
1. \*\*Learning Similarities and Differences:\*\* Siamese networks excel at learning to differentiate between pairs of inputs, which can be leveraged to learn a new representation space where similar items (regardless of sensitive attributes) are closer.  
2. \*\*Transferability of Learned Representations:\*\* The new representation space learned by a Siamese network can potentially be used for multiple tasks, including classification, making it versatile.  
3. \*\*Less Direct Conflict Between Objectives:\*\* Unlike adversarial training, where there's a direct opposition between the main and adversarial objectives, Siamese networks focus on learning representations, potentially leading to more stable training.  
  
\*\*Cons:\*\*  
1. \*\*Indirect Approach to Bias Mitigation:\*\* While Siamese networks can learn bias-free representations, the debiasing is more implicit compared to the direct adversarial approach, potentially making it harder to control or measure fairness.  
2. \*\*Dependency on Pairwise Data:\*\* Effective training of Siamese networks requires carefully selected pairs of data points, which can be challenging to construct, especially in terms of ensuring that pairs are meaningful for debiasing.  
3. \*\*Complexity in Evaluation:\*\* Evaluating the fairness of the learned representation space can be more complex, requiring additional steps or metrics beyond traditional classification accuracy or fairness measures.  
  
### Comparison and Considerations  
  
- \*\*Objective Alignment:\*\* If the primary goal is direct bias mitigation with measurable fairness criteria, adversarial NN might offer a more straightforward approach. In contrast, if the goal is to learn a generalizable, bias-free representation, a Siamese network could be more suitable.  
- \*\*Training Stability and Complexity:\*\* Adversarial training involves a delicate balance that can be hard to achieve, while Siamese networks, though requiring careful pair selection, might offer more stable training dynamics.  
- \*\*Application Scope:\*\* Adversarial NNs are tailored for debiasing by design and might be more effective in scenarios where fairness is a critical concern. Siamese networks offer a broader application range, potentially beneficial for tasks requiring similarity learning or transfer learning.  
  
In summary, the choice between an adversarial NN and a Siamese network for debiasing in binary classification depends on the specific requirements of the task, including the desired balance between accuracy and fairness, the nature of the available data, and the complexity of the training process that can be managed. Experimentation with both approaches could provide insights into which method best meets the needs of a particular application.

Explain final score and how debiasing must be done to optimise the training of the adversarial NN or siamese NN. Final score is calculated as follows :  
final\_score = (eval\_scores['macro\_fscore']+ (1-eval\_scores['TPR\_GAP']))/2  
  
refering to code bellow :  
  
import pandas as pd  
from sklearn import metrics  
from sklearn.utils.class\_weight import compute\_class\_weight  
from sklearn.utils import shuffle  
import numpy as np  
import json  
from collections import Counter  
  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import f1\_score  
from sklearn.metrics import recall\_score  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import mean\_absolute\_error  
from sklearn.metrics import mean\_squared\_error  
from sklearn.metrics import r2\_score  
import numpy as np  
from itertools import combinations  
from tqdm import tqdm  
  
from collections import defaultdict  
  
  
def confusion\_matrix\_based\_scores(cnf):  
  
"""Calculate confusion matrix based scores.  
Implementation from https://stackoverflow.com/a/43331484  
See https://en.wikipedia.org/wiki/Confusion\_matrix for different scores  
  
Args:  
cnf (np.array): a confusion matrix.  
  
Returns:  
dict: a set of metrics for each class, indexed by the metric name.  
  
"""  
  
FP = cnf.sum(axis=0) - np.diag(cnf) + 1e-5  
  
FN = cnf.sum(axis=1) - np.diag(cnf) + 1e-5  
  
TP = np.diag(cnf) + 1e-5  
  
TN = cnf.sum() - (FP + FN + TP) + 1e-5  
  
  
  
# Sensitivity, hit rate, recall, or true positive rate  
TPR = TP / (TP + FN)  
  
# Specificity or true negative rate  
TNR = TN / (TN + FP)  
  
# Precision or positive predictive value  
PPV = TP / (TP + FP)  
  
# Negative predictive value  
NPV = TN / (TN + FN)  
  
# Fall out or false positive rate  
FPR = FP / (FP + TN)  
  
# False negative rate  
FNR = FN / (TP + FN)  
  
# False discovery rate  
FDR = FP / (TP + FP)  
  
  
# Overall accuracy  
ACC = (TP + TN) / (TP + FP + FN + TN)  
  
  
# Positive Prediction Rates  
PPR = (TP + FP) / (TP + FP + FN + TN)  
  
  
return {  
"TPR": TPR,  
"TNR": TNR,  
"PPV": PPV,  
"NPV": NPV,  
"FPR": FPR,  
"FNR": FNR,  
"FDR": FDR,  
"ACC": ACC,  
"PPR": PPR,  
}  
  
  
  
  
  
def power\_mean(series, p, axis=0):  
  
"""calculate the generalized mean of a given list.  
  
Args:  
series (list): a list of numbers.  
p (int): power of the generalized mean aggregation  
axis (int, optional): aggregation along which dim of the input. Defaults to 0.  
  
Returns:  
np.array: aggregated scores.  
"""  
  
if p > 50:  
return np.max(series, axis=axis)  
  
elif p < -50:  
return np.min(series, axis=axis)  
  
else:  
total = np.mean(np.power(series, p), axis=axis)  
return np.power(total, 1 / p)  
  
  
  
  
  
def aggregation\_GAP(distinct\_groups, all\_scores, metric="TPR", group\_agg\_power=None, class\_agg\_power=2):  
  
"""Aggregate fairness metrics at the group level and class level.  
  
Args:  
distinct\_groups (list): a list of distinc labels of protected groups.  
all\_scores (dict): confusion matrix based scores for each protected group and all.  
metric (str, optional): fairness metric. Defaults to "TPR".  
group\_agg\_power (int, optional): generalized mean aggregation power at the group level. Use absolute value aggregation if None. Defaults to None.  
class\_agg\_power (int, optional): generalized mean aggregation power at the class level. Defaults to 2.  
  
Returns:  
np.array: aggregated fairness score.  
"""  
  
group\_scores = []  
  
for gid in distinct\_groups:  
  
# Save the TPR direct to the list  
group\_scores.append(all\_scores[gid][metric])  
  
# n\_class \* n\_groups  
  
scores = np.stack(group\_scores, axis=1)  
  
# Calculate GAP (n\_class \* n\_groups) - (n\_class \* 1)  
score\_gaps = scores - all\_scores["overall"][metric].reshape(-1, 1)  
  
# Sum over gaps of all protected groups within each class  
if group\_agg\_power is None:  
score\_gaps = np.sum(abs(score\_gaps), axis=1)  
  
else:  
score\_gaps = power\_mean(score\_gaps, p=group\_agg\_power, axis=1)  
  
# Aggregate gaps of each class, RMS by default  
score\_gaps = power\_mean(score\_gaps, class\_agg\_power)  
  
return score\_gaps  
  
  
def aggregation\_Ratio(distinct\_groups, all\_scores, metric="TPR", group\_agg\_power=None, class\_agg\_power=2):  
  
"""Aggregate fairness metric ratios at the group level and class level.  
  
Args:  
distinct\_groups (list): a list of distinc labels of protected groups.  
all\_scores (dict): confusion matrix based scores for each protected group and all.  
metric (str, optional): fairness metric. Defaults to "TPR".  
group\_agg\_power (int, optional): generalized mean aggregation power at the group level. Use absolute value aggregation if None. Defaults to None.  
class\_agg\_power (int, optional): generalized mean aggregation power at the class level. Defaults to 2.  
  
Returns:  
np.array: aggregated fairness score.  
  
"""  
  
group\_scores = []  
  
for gid in distinct\_groups:  
  
# Save the TPR direct to the list  
group\_scores.append(all\_scores[gid][metric])  
  
# n\_class \* n\_groups  
  
scores = np.stack(group\_scores, axis=1)  
  
# Calculate GAP (n\_class \* n\_groups) - (n\_class \* 1)  
score\_ratios = scores / all\_scores["overall"][metric].reshape(-1, 1)  
  
# Sum over ratios of all protected groups within each class  
  
if group\_agg\_power is None:  
score\_ratios = np.sum(abs(score\_ratios), axis=1)  
  
else:  
score\_ratios = power\_mean(score\_ratios, p=group\_agg\_power, axis=1)  
  
# Aggregate ratios of each class, RMS by default  
score\_ratios = power\_mean(score\_ratios, class\_agg\_power)  
  
return score\_ratios  
  
  
  
  
  
def gap\_eval\_scores(y\_pred, y\_true, protected\_attribute, metrics=["TPR", "FPR", "PPR"], args=None):  
  
"""fairness evaluation  
  
Args:  
y\_pred (np.array): model predictions.  
y\_true (np.array): target labels.  
protected\_attribute (np.array): protected labels.  
metrics (list, optional): a list of metric names that will be considered for fairness evaluation. Defaults to ["TPR","FPR","PPR"].  
  
Returns:  
tuple: (fairness evaluation results, confusion matrices)  
  
"""  
  
y\_pred = np.array(y\_pred)  
y\_true = np.array(y\_true)  
protected\_attribute = np.array(protected\_attribute)  
  
  
if (args is not None) and args.regression:  
  
eval\_scores = {  
"mean\_absolute\_error": mean\_absolute\_error(y\_true, y\_pred),  
"mean\_squared\_error": mean\_squared\_error(y\_true, y\_pred),  
"r2\_score": r2\_score(y\_true, y\_pred),  
}  
  
# Processing regression labels for fairness evaluation under the classification framework  
y\_true = pd.cut(np.squeeze(y\_true), bins=args.regression\_bins, labels=False, duplicates="drop")  
y\_pred = pd.cut(np.squeeze(y\_pred), bins=args.regression\_bins, labels=False, duplicates="drop")  
y\_true = np.nan\_to\_num(y\_true, nan=0)  
y\_pred = np.nan\_to\_num(y\_pred, nan=0)  
  
else:  
  
# performance evaluation  
eval\_scores = {  
"accuracy": accuracy\_score(y\_true, y\_pred),  
"macro\_fscore": f1\_score(y\_true, y\_pred, average="macro"),  
"micro\_fscore": f1\_score(y\_true, y\_pred, average="micro"),  
}  
  
all\_scores = {}  
confusion\_matrices = {}  
  
# Overall evaluation  
distinct\_labels = [i for i in range(len(set(y\_true)))]  
overall\_confusion\_matrix = confusion\_matrix(y\_true=y\_true, y\_pred=y\_pred, labels=distinct\_labels)  
confusion\_matrices["overall"] = overall\_confusion\_matrix  
all\_scores["overall"] = confusion\_matrix\_based\_scores(overall\_confusion\_matrix)  
  
# Group scores  
distinct\_groups = [i for i in range(len(set(protected\_attribute)))]  
  
for gid in distinct\_groups:  
group\_identifier = (protected\_attribute == gid)  
group\_confusion\_matrix = confusion\_matrix(y\_true=y\_true[group\_identifier], y\_pred=y\_pred[group\_identifier],  
labels=distinct\_labels)  
confusion\_matrices[gid] = group\_confusion\_matrix  
all\_scores[gid] = confusion\_matrix\_based\_scores(group\_confusion\_matrix)  
  
for \_metric in metrics:  
eval\_scores["{}\_GAP".format(\_metric)] = aggregation\_GAP(distinct\_groups=distinct\_groups, all\_scores=all\_scores,  
metric=\_metric)  
  
return eval\_scores, confusion\_matrices

**EagleMind**

EagleMind> 5: The final score calculation and the optimization of adversarial or Siamese neural networks (NN) for debiasing hinge on understanding how to balance accuracy with fairness. The final score formula provided:  
  
**python**

final\_score = (eval\_scores['macro\_fscore']+ (1-eval\_scores['TPR\_GAP']))/2

combines two critical aspects: the Macro F1 score, which measures the accuracy of the model across different classes, and the True Positive Rate (TPR) Gap, which measures fairness in terms of the model's performance across different groups defined by a sensitive attribute.  
  
### Understanding the Final Score Components  
  
1. \*\*Macro F1 Score\*\*: Averages the F1 scores of all classes, balancing precision and recall. It's a robust metric for multi-class classification problems, especially when class imbalances exist.  
  
2. \*\*TPR Gap (True Positive Rate Gap)\*\*: Measures the difference in TPR (recall for the positive class) between groups defined by the sensitive attribute. A smaller TPR Gap indicates less bias, as it means the model's ability to correctly identify positive instances is similar across groups.  
  
### Adversarial NN for Debiasing  
  
\*\*Pros:\*\*  
- Directly targets bias by training the model to perform well on the primary task while making the sensitive attribute difficult to predict.  
- Flexible and can be adapted to various data types and model architectures.  
  
\*\*Cons:\*\*  
- Training can be challenging due to the adversarial setup, requiring careful tuning of the model and training process.  
- May lead to decreased performance on the primary task if the debiasing overly constrains the model.  
  
\*\*Optimization Strategies:\*\*  
- \*\*Loss Weight Tuning\*\*: Adjust the weight of the adversarial loss relative to the main task loss to find a balance that reduces bias without significantly harming the primary task performance.  
- \*\*Adversarial Training Scheduling\*\*: Introduce the adversarial training gradually or apply it more aggressively once the primary model has achieved a certain level of performance.  
  
### Siamese NN for Debiasing  
  
\*\*Pros:\*\*  
- Learns embeddings that are useful for distinguishing between classes while minimizing the influence of the sensitive attribute.  
- Can be used to preprocess data, creating representations that downstream models can use without inheriting bias.  
  
\*\*Cons:\*\*  
- Indirectly addresses bias, which might make it harder to measure and control debiasing effectiveness.  
- Requires careful selection of pairs and triplets for training, which can be complex and data-intensive.  
  
\*\*Optimization Strategies:\*\*  
- \*\*Contrastive Loss Adjustment\*\*: Modify the contrastive loss function to encourage embeddings to be similar for the same class regardless of the sensitive attribute and dissimilar for different classes.  
- \*\*Triplet Selection\*\*: Focus on selecting challenging triplets that help the model learn meaningful distinctions that are not based on the sensitive attribute.  
  
### Balancing Accuracy and Fairness for Final Score Optimization  
  
To optimize the training of either adversarial or Siamese NNs for the best final score, consider the following:  
  
- \*\*Evaluation and Feedback Loop\*\*: Continuously evaluate both the Macro F1 score and the TPR Gap during training. Use these evaluations to adjust the training process dynamically.  
- \*\*Fairness-Aware Hyperparameter Tuning\*\*: Experiment with different hyperparameters, focusing on those that influence the balance between accuracy and fairness, such as the learning rate, loss weights, and the architecture of the adversarial or Siamese components.  
- \*\*Data Augmentation and Resampling\*\*: Use data augmentation or resampling techniques to address class imbalances that might affect both accuracy and fairness.  
  
Ultimately, the choice between adversarial and Siamese NNs, and the specific strategies for optimizing them, should be guided by the characteristics of the dataset, the nature of the sensitive attribute, and the specific requirements for accuracy and fairness in the application context.

the dataset is an 576 embedding from bert + label (from 0 to 27) + 1/0 to bellonging to sensitive class

**EagleMind**

EagleMind> 6: Given the dataset comprises 576-dimensional embeddings from BERT, labels ranging from 0 to 27, and a binary indicator for belonging to a sensitive class, let's delve into how you can approach training both adversarial and Siamese neural networks to optimize for the final score, considering the dataset specifics.  
  
### Adversarial Neural Network Approach  
  
Given the dataset specifics, an adversarial neural network can be structured to ensure the model's predictions are both accurate and fair, with respect to the sensitive attribute.  
  
\*\*Model Design:\*\*  
- \*\*Primary Classifier\*\*: This component focuses on predicting the label (0 to 27) based on the 576-dimensional BERT embeddings. Given the high dimensionality and complexity, a deep neural network with multiple layers can be utilized to capture the intricate patterns in the embeddings.  
- \*\*Adversary\*\*: This component attempts to predict the sensitive attribute (1/0) from the primary classifier's representations. The goal during training is to minimize the adversary's accuracy, indicating that the sensitive attribute cannot be inferred from the model's predictions, thus ensuring fairness.  
  
\*\*Training Strategy:\*\*  
- Initially, train the primary classifier on the task of predicting the labels while not exposing it to the sensitive attribute directly.  
- Introduce the adversary, trained to predict the sensitive attribute from the primary classifier's output or intermediate representations.  
- Adjust the training process by applying a gradient reversal layer or modifying the loss function to penalize the model when the adversary successfully predicts the sensitive attribute, encouraging the primary classifier to learn representations that are invariant to the sensitive attribute.  
  
### Siamese Neural Network Approach  
  
For the Siamese network, the focus shifts to learning embeddings that are useful for the classification task while minimizing the influence of the sensitive attribute.  
  
\*\*Model Design:\*\*  
- \*\*Siamese Architecture\*\*: Utilize two identical neural networks that accept a pair of inputs (two 576-dimensional BERT embeddings). The networks learn to project these inputs into a space where embeddings of the same label are closer together, regardless of the sensitive attribute.  
- \*\*Contrastive Loss\*\*: This loss function encourages the model to minimize the distance between embeddings of the same class and maximize the distance between embeddings of different classes.  
  
\*\*Training Strategy:\*\*  
- Generate pairs or triplets of data points considering the labels and the sensitive attribute. For example, positive pairs can be from the same class but different sensitive attributes, encouraging the model to learn embeddings that are similar for the same class regardless of the sensitive attribute.  
- Train the Siamese network to optimize the contrastive loss, adjusting the embeddings to be useful for classification while being fair concerning the sensitive attribute.  
  
### Balancing Accuracy and Fairness  
  
For both approaches, the final score calculation:  
  
**python**

final\_score = (eval\_scores['macro\_fscore']+ (1-eval\_scores['TPR\_GAP']))/2

emphasizes the need to balance accuracy (Macro F1 score) with fairness (1 - TPR Gap). To optimize training:  
  
- \*\*Regular Evaluation\*\*: Continuously evaluate both the Macro F1 score and the TPR Gap during training. Use these metrics to guide adjustments in the model architecture, loss functions, and training strategy.  
- \*\*Fairness-Aware Hyperparameter Tuning\*\*: Experiment with hyperparameters that influence the balance between accuracy and fairness. This includes learning rates, loss weights, and the architecture of specific components like the adversary in adversarial training or the embedding layers in Siamese networks.  
- \*\*Data Augmentation\*\*: Consider augmenting the dataset to balance the representation of the sensitive attribute across classes, which can help in reducing the TPR Gap.  
  
### Implementation Considerations  
  
- \*\*Computational Resources\*\*: Both adversarial and Siamese networks can be computationally intensive, especially with high-dimensional embeddings and complex architectures. Ensure adequate computational resources for training and experimentation.  
- \*\*Hyperparameter Sensitivity\*\*: Both models might be sensitive to hyperparameter settings, requiring extensive experimentation to find the optimal configuration for your specific dataset.  
  
By carefully designing, training, and evaluating your model, you can develop a system that performs well on the classification task while also being fair with respect to the sensitive attribute, optimizing for the final score as defined.

I have the following issue with the approch you suggest. the models are being trained or overall sample accuracy, where as the evaluation fuction provided above, evaluated the average of accuracy for each class. Re-evaluate suggested approach to adapt accuracy score for training the model, on my specific evaluation function : final\_score = (eval\_scores['macro\_fscore']+ (1-eval\_scores['TPR\_GAP']))/2

**EagleMind**

EagleMind> 7: Given the specific evaluation function where the final score is calculated as the average of the Macro F1 score and the complement of the True Positive Rate (TPR) Gap, it's crucial to adapt the training approach to align more closely with this evaluation criterion. The challenge lies in optimizing for both class-wise accuracy (as captured by the Macro F1 score) and fairness (as measured by the TPR Gap), rather than overall sample accuracy. Here's a revised approach considering this nuanced evaluation function:  
  
### Training with a Focus on Macro F1 Score  
  
1. \*\*Macro F1 Optimization\*\*: Since the Macro F1 score is an arithmetic mean of class-wise F1 scores, it inherently values performance across all classes equally, making it a suitable objective when class imbalances exist. To optimize for the Macro F1 score during training:  
- \*\*Custom Loss Function\*\*: Implement a custom loss function that directly or indirectly optimizes for the Macro F1 score. This could involve designing a differentiable approximation of the F1 score that can be used as a training objective.  
- \*\*Class Weights\*\*: Use class weights to address class imbalances. Compute class weights inversely proportional to class frequencies and apply these during model training to place more emphasis on underrepresented classes.  
  
2. \*\*Model Architecture Adjustments\*\*: Ensure the model architecture is capable of capturing the nuances of each class. This might involve increasing model capacity or incorporating mechanisms like attention to focus on relevant parts of the input embeddings.  
  
### Incorporating Fairness into Training  
  
1. \*\*Fairness-aware Training\*\*: To minimize the TPR Gap, integrate fairness considerations directly into the training process:  
- \*\*Adversarial Training for Fairness\*\*: Continue with the adversarial training approach, but adjust the adversary to focus on minimizing the TPR Gap specifically. This could involve training the adversary to predict the TPR for each class or group and penalizing discrepancies.  
- \*\*Fairness Constraints\*\*: Incorporate fairness constraints into the model's loss function. For example, add a term to the loss that penalizes the model for large TPR Gaps between groups defined by the sensitive attribute.  
  
### Evaluation and Adjustment Loop  
  
1. \*\*Continuous Evaluation\*\*: Regularly evaluate the model on both the Macro F1 score and the TPR Gap during training. Use a validation set that mirrors the class distribution and sensitive attribute distribution of the training set.  
2. \*\*Hyperparameter Tuning\*\*: Experiment with hyperparameters that influence the balance between accuracy (Macro F1) and fairness (TPR Gap). This includes learning rates, the strength of fairness regularization, and the architecture of the adversarial component if used.  
  
### Practical Considerations  
  
- \*\*Evaluation Metric in Training Loop\*\*: Directly optimizing for the final score during training might not be straightforward due to the non-differentiable nature of the F1 score and the complexity of calculating the TPR Gap. Instead, focus on optimizing components of the final score (like the Macro F1 score) and use the full evaluation function primarily for validation and testing.  
- \*\*Fairness-aware Data Augmentation\*\*: Consider data augmentation techniques that can help balance the representation of the sensitive attribute across classes, potentially reducing the TPR Gap without explicitly modifying the model architecture or loss function.  
  
### Final Thoughts  
  
Optimizing for a combined metric of accuracy (Macro F1) and fairness (1 - TPR Gap) requires a nuanced approach that balances these sometimes competing objectives. By focusing on class-wise performance and integrating fairness directly into the training process, you can develop a model that performs well according to the specified evaluation function. Remember, achieving the optimal balance may require iterative experimentation with model architectures, loss functions, and training strategies.

code the customed loss function for the adversarial NN and provide the full code to train the adversarial NN (based on all information allready provided above (including evaluation functions) and doing the post processing debiasing.

**EagleMind**

EagleMind> 8: Crafting a custom loss function for an adversarial neural network (NN) that incorporates both the Macro F1 score and fairness considerations (via the TPR Gap) is complex due to the non-differentiability of these metrics. However, I'll guide you through an approach that approximates this goal and integrates post-processing debiasing. Note that this solution will be conceptual and may require adjustments for practical implementation.  
  
### Step 1: Define the Custom Loss Function  
  
This custom loss function aims to optimize the primary model for accuracy while penalizing unfairness detected by the adversary. Since the Macro F1 score and TPR Gap are not directly differentiable, we'll use proxy metrics that are differentiable for training.  
  
**python**

import tensorflow as tf

def custom\_loss(y\_true, y\_pred, alpha=0.5):

"""

A custom loss function that combines cross-entropy for accuracy and mean squared error as a proxy for fairness.

Args:

- y\_true: true labels

- y\_pred: predicted labels

- alpha: weighting factor to balance accuracy and fairness

"""

# Cross-entropy for the accuracy part

cross\_entropy = tf.keras.losses.sparse\_categorical\_crossentropy(y\_true, y\_pred)

# Mean squared error as a proxy for fairness

mse = tf.keras.losses.mean\_squared\_error(y\_true, y\_pred)

# Combine the losses

combined\_loss = alpha \* cross\_entropy + (1 - alpha) \* mse

return combined\_loss

### Step 2: Build the Adversarial NN Model  
  
We'll construct a simple adversarial model with TensorFlow/Keras. This model has a primary classifier and an adversary designed to predict the sensitive attribute.  
  
**python**

from tensorflow.keras.layers import Input, Dense, Dropout

from tensorflow.keras.models import Model

def build\_adversarial\_model(input\_dim, num\_classes):

# Primary task model

inputs = Input(shape=(input\_dim,))

x = Dense(128, activation='relu')(inputs)

x = Dropout(0.5)(x)

primary\_output = Dense(num\_classes, activation='softmax', name='primary\_output')(x)

# Adversary model

adversary\_output = Dense(1, activation='sigmoid', name='adversary\_output')(x)

model = Model(inputs=inputs, outputs=[primary\_output, adversary\_output])

# Compile the model

model.compile(optimizer='adam',

loss={'primary\_output': custom\_loss, 'adversary\_output': 'binary\_crossentropy'},

metrics={'primary\_output': 'accuracy', 'adversary\_output': 'accuracy'})

return model

### Step 3: Train the Adversarial NN  
  
Assuming `X\_train`, `Y\_train`, and `S\_train` are your features, target labels, and sensitive attributes respectively:  
  
**python**

model = build\_adversarial\_model(input\_dim=576, num\_classes=28)

# Train the model

model.fit(X\_train, {'primary\_output': Y\_train, 'adversary\_output': S\_train},

batch\_size=32, epochs=10, validation\_split=0.2)

### Step 4: Post-processing for Debiasing  
  
After training, apply a post-processing step to adjust predictions and further mitigate bias. This step can be based on the Equalized Odds post-processing method or any other suitable technique.  
  
**python**

# Placeholder for post-processing step

# This step would adjust the model's predictions based on the sensitive attribute to ensure fairness

# Actual implementation would depend on the chosen post-processing technique

### Note:  
  
- The custom loss function provided is a simplified proxy for the actual evaluation metrics. Direct optimization for Macro F1 and fairness (TPR Gap) in a differentiable manner is challenging and often requires approximations or alternative strategies.  
- The adversarial model architecture and training procedure are conceptual. Depending on your dataset and specific task requirements, you may need to adjust the model architecture, loss functions, and training hyperparameters.  
- Post-processing debiasing techniques require careful calibration and validation to ensure they effectively reduce bias without unduly compromising the model's accuracy.  
  
This framework provides a starting point for building an adversarial neural network aimed at balancing accuracy with fairness. Experimentation and iterative refinement will be key to optimizing performance according to your specific evaluation criteria.

can you give me an other implementation of step 1 for the cutom loss function. Do not include weights for each category, but include averaging the accuracy of each class. Provide python code with rationnal

**EagleMind**

EagleMind> 9: Implementing a custom loss function that directly optimizes for class-wise accuracy without explicitly including weights for each category, but instead averages the accuracy of each class, presents a unique challenge. This is because traditional loss functions like cross-entropy are not designed to directly optimize for accuracy, especially in a class-wise manner. However, we can approximate this goal by focusing on minimizing the class-wise cross-entropy loss, which indirectly optimizes for accuracy.  
  
Given the constraints of differentiable loss functions, a practical approach is to ensure that the loss for each class contributes equally to the total loss, mimicking the effect of optimizing for class-wise accuracy. Here's an implementation that attempts to address this:  
  
### Step 1: Custom Loss Function Implementation  
  
This custom loss function will calculate the cross-entropy loss for each class individually and then average these losses. This approach ensures that each class contributes equally to the total loss, aligning with the goal of averaging class-wise accuracy.  
  
**python**

import tensorflow as tf

def classwise\_loss(y\_true, y\_pred):

"""

Custom loss function that calculates the cross-entropy loss for each class

individually and then averages these losses.

Args:

- y\_true: Tensor of true labels, one-hot encoded.

- y\_pred: Tensor of predicted labels, as probabilities.

Returns:

- loss: A scalar tensor representing the average class-wise cross-entropy loss.

"""

# Ensure predictions sum to 1

y\_pred = tf.clip\_by\_value(y\_pred, tf.keras.backend.epsilon(), 1 - tf.keras.backend.epsilon())

y\_pred = tf.math.divide(y\_pred, tf.reduce\_sum(y\_pred, axis=-1, keepdims=True))

# Calculate cross-entropy loss for each class

classwise\_losses = -tf.reduce\_sum(y\_true \* tf.math.log(y\_pred), axis=0)

# Average the class-wise losses

loss = tf.reduce\_mean(classwise\_losses)

return loss

### Rationale:  
  
- \*\*Cross-Entropy for Each Class\*\*: By calculating the cross-entropy loss for each class individually, we ensure that the model's performance on each class affects the total loss. This is crucial for tasks with imbalanced datasets where certain classes might be underrepresented.  
- \*\*Averaging Class-wise Losses\*\*: Averaging these class-wise losses aligns the training objective more closely with the Macro F1 score component of the final score. It ensures that improving the model's performance on any single class will have a proportional impact on reducing the overall loss, encouraging the model to perform well across all classes.  
- \*\*Clipping and Normalization\*\*: Clipping `y\_pred` and ensuring it sums to 1 prevents numerical instability in the logarithm calculation and ensures that `y\_pred` represents valid probabilities.  
  
### Integration with Model Training:  
  
When integrating this loss function into model training, ensure that your `y\_true` labels are one-hot encoded, as the loss function expects probabilities for each class.  
  
**python**

from tensorflow.keras.utils import to\_categorical

# Assuming num\_classes is defined and represents the total number of classes

y\_train\_onehot = to\_categorical(Y\_train, num\_classes=num\_classes)

# Compile the model with the custom loss function

model.compile(optimizer='adam', loss=classwise\_loss, metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train\_onehot, batch\_size=32, epochs=10, validation\_split=0.2)

This approach to designing a custom loss function aims to align the training process more closely with the evaluation metric, especially the Macro F1 score component, by emphasizing class-wise performance. However, it's important to note that direct optimization for accuracy and fairness metrics, especially those involving non-differentiable components like the F1 score, often requires creative approximations or multi-stage training processes.

does it make sense to include fairness in the custom loss function, whereas the adversarial NN specifically aims to remove fairness in the representation space generated by the main model (classifier)

**EagleMind**

EagleMind> 10: Integrating fairness directly into the custom loss function of the main model (classifier) in an adversarial neural network setup raises an important consideration regarding the design and objectives of adversarial training. The primary goal of adversarial training is to learn a representation that performs well on the main task while ensuring that this representation does not allow for accurate prediction of the sensitive attribute, thereby promoting fairness. Let's dissect the implications and rationale behind including or excluding fairness directly in the custom loss function of the classifier.  
  
### Adversarial Training Objective  
  
- \*\*Primary Model (Classifier)\*\*: Aims to accurately predict the target labels from the input data.  
- \*\*Adversary Model\*\*: Aims to predict the sensitive attribute from the representations learned by the primary model. The better the adversary at predicting the sensitive attribute, the more biased the representation is considered.  
  
### Including Fairness in the Classifier's Loss Function  
  
\*\*Pros\*\*:  
1. \*\*Direct Optimization for Fairness\*\*: Including a fairness term in the classifier's loss function allows for direct optimization towards a fairer model. This can lead to quicker convergence towards fair representations without relying solely on the adversarial component.  
2. \*\*Simpler Training Dynamics\*\*: It may simplify the training dynamics by reducing the reliance on the adversarial component to enforce fairness, potentially leading to more stable training.  
  
\*\*Cons\*\*:  
1. \*\*Conflicting Objectives\*\*: The primary model's objective to minimize its own loss (including the fairness term) might conflict with the adversarial training objective. This could potentially weaken the adversarial signal meant to guide the model towards fairness.  
2. \*\*Complexity in Loss Design\*\*: Designing a loss function that effectively balances accuracy and fairness within the classifier can be challenging. It requires careful consideration to ensure that the fairness term does not dominate or undermine the accuracy objective.  
  
### Excluding Fairness from the Classifier's Loss Function  
  
\*\*Pros\*\*:  
1. \*\*Clear Separation of Objectives\*\*: Keeping the classifier's loss function focused on accuracy allows for a clear separation of objectives. The classifier focuses on accuracy, while the adversary focuses on fairness. This separation can make the adversarial training process more interpretable.  
2. \*\*Stronger Adversarial Signal\*\*: Excluding fairness from the classifier's loss ensures that the adversarial signal remains the primary mechanism for promoting fairness. This can lead to more effective adversarial training, as the classifier and adversary engage in a clearer optimization "battle."  
  
\*\*Cons\*\*:  
1. \*\*Potential for Slower Convergence to Fairness\*\*: Relying solely on the adversarial component to enforce fairness might result in slower convergence towards fair representations, as the classifier initially learns representations without considering fairness.  
2. \*\*Increased Training Complexity\*\*: The adversarial training process can be more complex and harder to stabilize, requiring careful tuning of the adversarial and classifier components.  
  
### Conclusion  
  
Whether to include fairness directly in the classifier's loss function in an adversarial setup depends on the specific goals, dataset characteristics, and the desired balance between accuracy and fairness. Both approaches have their merits and potential drawbacks.  
  
- \*\*For tasks where fairness is as critical as accuracy\*\*, incorporating a fairness term in the classifier's loss might provide a more direct path to achieving both objectives.  
- \*\*For tasks where the primary goal is accuracy, but fairness cannot be ignored\*\*, relying on the adversarial component to enforce fairness allows the classifier to focus on accuracy while still moving towards fair representations.  
  
Experimentation and validation are key to determining the most effective approach for a given application, considering the trade-offs between direct optimization for fairness and the dynamics of adversarial training.

 ⬆ ⬇

Haut du formulaire

Bas du formulaire