```
# !pip install matplotlib pandas transformers seaborn
```

Проект по анализу данных "Зависимость результатов multiple-choice классификации от семантической близости классов"

Гипотеза

H0: Семантическая близость классов обратно пропорциональна precision LLM классификатора. H1: Семантическая близость классов пропорциональна precision LLM классификатора.

Данные

Данные взяты из бенчмарка дискурсивных явлений DISRP (русскоязычный сабсет). Перед моделью стоит задача классификации дискурсивных отношений между предложениями (частями предложений). Всего в датасете 22 класса. Для сравнения семантической близости классов использовался ruBERT-base-cased.

Методология

Для проверки гипотезы мы нашли для каждого класса в датасете DISRP 4 ближайших класс по семантической близости и 4 самых дальних класса. Таким образом мы собрали 4 варианта датасета, различающихся только набором классов подаваемых модели на вход для выбора: 5 ближайших классов, 5 дальних классов, 5 случайных классов (таргетный включен) и 22 класса (все классы). Мы проскорили модель gpt-4o-mini на каждом из этих датасетов и сравнили precision для каждого из них, а также посчитали корреляцию между precision и семантической близостью классов внутри сабсета.

```
import json
import random
import numpy as np
import pandas as pd
from tqdm import tqdm
from transformers import pipeline
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.preprocessing import normalize
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import (
    accuracy_score,
    precision_recall_fscore_support,
    confusion_matrix,
    classification_report,
```

```
)
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import pearsonr
```

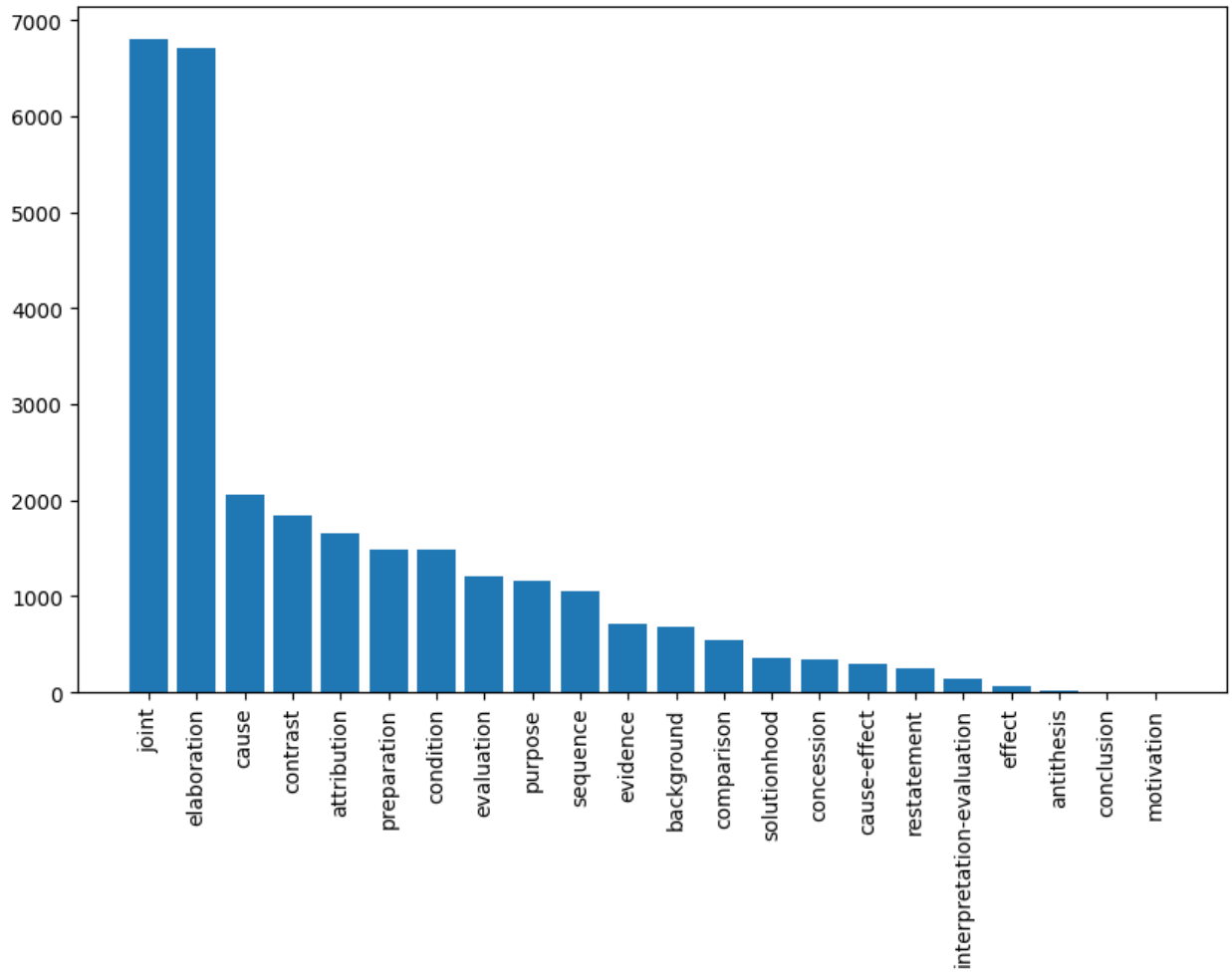
Data loading

```
with open("full_disrpt.json", "r", encoding="utf-8") as f:
    data = json.load(f)

embedder = pipeline("feature-extraction", model="DeepPavlov/rubert-
base-cased")


def get_embedding(text):
    """
    Get the embedding for a given text using the embedder pipeline.
    """
    embedding = embedder(text)
    return np.mean(embedding[0], axis=0)
```

```
Some weights of the model checkpoint at DeepPavlov/rubert-base-cased
were not used when initializing BertModel: ['cls.predictions.bias',
'cls.predictions.decoder.bias', 'cls.predictions.decoder.weight',
'cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.dense.bias',
'cls.predictions.transform.dense.weight', 'cls.seq_relationship.bias',
'cls.seq_relationship.weight']
- This IS expected if you are initializing BertModel from the
checkpoint of a model trained on another task or with another
architecture (e.g. initializing a BertForSequenceClassification model
from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the
checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Device set to use cuda:0
```

```
df = pd.DataFrame(columns=["start", "end", "label"])
for item in data.values():
    row = {
        # '_id': item['id'],
        "start": item["sent_1"],
        "end": item["sent_2"],
        "label": item["label"],
    }
    df.loc[-1] = row
    df.index = df.index + 1
df.head()
```

```
                                                      start  \
28867  - формируются коэффициенты модулей начальной и...
28866  В этой статье решено привести обобщение алгори...
28865                                Кабардинец везёт хлеб
28864  ##### — Какие именно из всего семейства лиценз...
28863  Помимо правильного рабочего места , о котором ...

                                                        end         label

28867                                    [ формула ] ;   elaboration

28866  которые могут быть описаны одной или нескольки...   elaboration

28865                                - едет в Подольск ,         joint

28864  ##### — Проекты фонда « Викимедиа » , лицензир...  solutionhood

28863  Регулярное выполнение асан поможет сохранить з...   elaboration


df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 28868 entries, 28867 to 0
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   start   28868 non-null  object
 1   end     28868 non-null  object
 2   label   28868 non-null  object
dtypes: object(3)
memory usage: 902.1+ KB

labels = df["label"].unique()
print(f"{len(labels)} Labels: {labels}")

22 Labels: ['elaboration' 'joint' 'solutionhood' 'cause' 'contrast'
'sequence'
 'purpose' 'preparation' 'concession' 'restatement' 'attribution'
 'evidence' 'evaluation' 'condition' 'background' 'cause-effect'
 'comparison' 'interpretation-evaluation' 'effect' 'antithesis'
 'conclusion' 'motivation']

plt.figure(figsize=(10, 6))
labels_amount = df["label"].value_counts()
plt.xticks(rotation=90)
plt.bar(labels_amount.index, labels_amount.values)

<BarContainer object of 22 artists>
```



Как можно заметить, дисбаланс классов в датасете достаточно большой, однако, для наших целей это не критично, т.к. мы не будем ничего обучать, а просто посмотрим на то, как ведут себя модели с различной комбинацией вариантов ответа. Всё же, нам нужно будет выделить сабсет учитывающий дисбаланс классов.

```
# Create a test set for scoring
test_set = df.sample(frac=0.1, random_state=5)
print(f"Test set size: {len(test_set)}")
print(f"{len(test_set['label'].unique())} labels in the test set")
print(test_set["label"].value_counts())
```

```
Test set size: 2887
22 labels in the test set
label
joint                          701
elaboration                    648
cause                          209
contrast                       187
attribution                    182
preparation                    166
```

```
condition                    149
purpose                      117
evaluation                   116
sequence                      98
evidence                      69
background                    66
comparison                    42
concession                    34
solutionhood                  31
cause-effect                  29
restatement                   23
interpretation-evaluation     14
effect                         3
conclusion                     1
antithesis                     1
motivation                     1
Name: count, dtype: int64
```

```
test_set.head()
```

```
                                                  start  \
6542           что если метафора на самом деле мертвая ,
21707                Полагаем , что первейшее условие ,
3770   вследствие чего он обладает средствами лингвис...
15858  « устраиваемая на звериных тропах большая запа...
22722       где я в крайнем случае и не помешаю никому ,

                                                    end       label
6542           она теряет свои композиционные свойства ,   condition
21707  В подавляющем большинстве недостатки произноше...       cause
3770                             является анализ того ,       joint
15858  - в значении « ловушка для рябчиков и куропато...  comparison
22722                                   если упаду ,   condition
```

Similar or contrasting labels

Теперь мы можем попробовать найти для каждого класса наиболее и наименее похожие на него классы. Для этого мы используем два разных подхода:

1. **Сравнение векторов классов**: мы можем сравнить векторы классов (самих строк) и найти наиболее похожие и непохожие классы по косинусной близости.
2. **Сравнение вручную**: мы можем сравнить классы полагаясь на собственное лингвистическое чутье и потом сравнить результаты с результатами первого подхода.

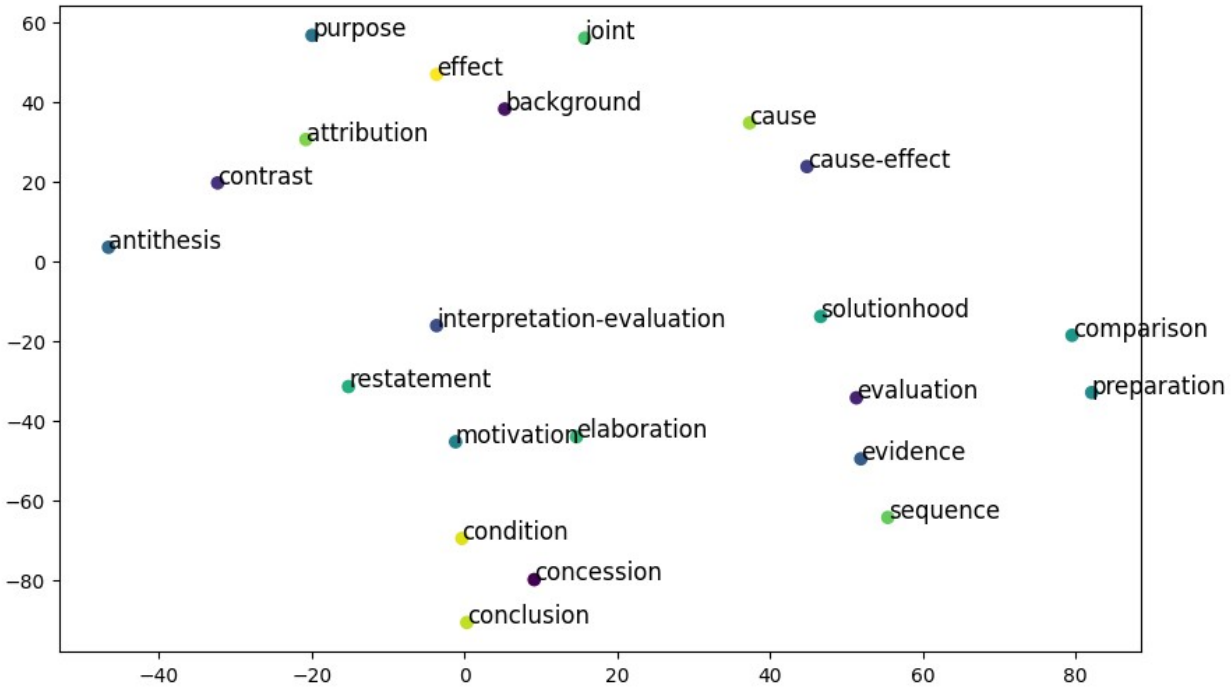
```
label_embeddings = []
for label in labels:
    embedding = get_embedding(label)
    label_embeddings.append({"label": label, "embedding": embedding})
```

```
labels_emb_df = pd.DataFrame(label_embeddings)
labels_emb_df.to_csv("label_embeddings_rubert.csv", index=False)
```

```
You seem to be using the pipelines sequentially on GPU. In order to
maximize efficiency please use a dataset
```

```
# Clustering labels
label_embeddings = np.array(labels_emb_df["embedding"].tolist())
label_embeddings = normalize(label_embeddings)

k = len(labels)  # Number of clusters equal to number of unique labels
kmeans = KMeans(n_clusters=k, random_state=42).fit(label_embeddings)

# Reduce dimensions for visualization
tsne = TSNE(n_components=2, random_state=42, perplexity=5)
label_embeddings = tsne.fit_transform(label_embeddings)

# Plotting the clusters with names
plt.figure(figsize=(10, 6))
for i, label in enumerate(labels):
    plt.annotate(label, (label_embeddings[i, 0], label_embeddings[i,
1]), fontsize=12)
plt.scatter(
    label_embeddings[:, 0],
    label_embeddings[:, 1],
    c=kmeans.labels_,
    cmap="viridis",
    marker="o",
)
```

```
<matplotlib.collections.PathCollection at 0x7d9ff3e65700>
```




```python
# Get top 4 closest labels to each label and top 4 furthest labels by cosine similarity

labels_dictionaty = {}
cos_sim_matrix = cosine_similarity(label_embeddings, label_embeddings)
for i, label in enumerate(labels):
    # Exclude the label itself by setting its similarity to -inf/+inf
    sim_scores = cos_sim_matrix[i].copy()
    sim_scores[i] = -np.inf  # for closest
    closest_idx = np.argsort(sim_scores)[-4:][::-1]
    sim_scores[i] = np.inf  # for furthest
    furthest_idx = np.argsort(sim_scores)[:4]
    labels_dictionaty[label] = {
        "closest": labels[closest_idx].tolist(),
        "furthest": labels[furthest_idx].tolist(),
    }
for label, values in labels_dictionaty.items():
    print(f"Label: {label}")
    print(f"  Closest: {values['closest']}")
    print(f"  Furthest: {values['furthest']}")
```

```
Label: elaboration
  Closest: ['concession', 'conclusion', 'condition', 'motivation']
  Furthest: ['purpose', 'effect', 'attribution', 'background']
Label: joint
  Closest: ['background', 'effect', 'cause', 'purpose']
  Furthest: ['interpretation-evaluation', 'restatement', 'motivation',
'condition']
```

```
Label: solutionhood
  Closest: ['comparison', 'preparation', 'evaluation', 'evidence']
  Furthest: ['antithesis', 'contrast', 'attribution', 'purpose']
Label: cause
  Closest: ['cause-effect', 'joint', 'background', 'effect']
  Furthest: ['restatement', 'interpretation-evaluation', 'motivation',
'condition']
Label: contrast
  Closest: ['attribution', 'antithesis', 'purpose', 'effect']
  Furthest: ['evaluation', 'preparation', 'evidence', 'solutionhood']
Label: sequence
  Closest: ['evidence', 'evaluation', 'elaboration', 'preparation']
  Furthest: ['attribution', 'contrast', 'purpose', 'effect']
Label: purpose
  Closest: ['attribution', 'effect', 'background', 'joint']
  Furthest: ['elaboration', 'concession', 'conclusion', 'condition']
Label: preparation
  Closest: ['solutionhood', 'comparison', 'evaluation', 'evidence']
  Furthest: ['contrast', 'antithesis', 'attribution', 'purpose']
Label: concession
  Closest: ['conclusion', 'condition', 'motivation', 'elaboration']
  Furthest: ['effect', 'purpose', 'background', 'joint']
Label: restatement
  Closest: ['interpretation-evaluation', 'motivation', 'condition',
'conclusion']
  Furthest: ['joint', 'background', 'cause', 'effect']
Label: attribution
  Closest: ['purpose', 'contrast', 'effect', 'background']
  Furthest: ['sequence', 'evidence', 'elaboration', 'evaluation']
Label: evidence
  Closest: ['sequence', 'evaluation', 'preparation', 'solutionhood']
  Furthest: ['attribution', 'contrast', 'purpose', 'antithesis']
Label: evaluation
  Closest: ['evidence', 'preparation', 'sequence', 'solutionhood']
  Furthest: ['contrast', 'attribution', 'antithesis', 'purpose']
Label: condition
  Closest: ['conclusion', 'motivation', 'concession', 'interpretation-
evaluation']
  Furthest: ['effect', 'background', 'joint', 'purpose']
Label: background
  Closest: ['joint', 'effect', 'purpose', 'cause']
  Furthest: ['interpretation-evaluation', 'motivation', 'condition',
'conclusion']
Label: cause-effect
  Closest: ['cause', 'comparison', 'solutionhood', 'joint']
  Furthest: ['antithesis', 'restatement', 'interpretation-evaluation',
'contrast']
Label: comparison
  Closest: ['solutionhood', 'preparation', 'evaluation', 'evidence']
```

```
  Furthest: ['antithesis', 'contrast', 'attribution', 'purpose']
Label: interpretation-evaluation
  Closest: ['motivation', 'condition', 'conclusion', 'restatement']
  Furthest: ['joint', 'background', 'effect', 'purpose']
Label: effect
  Closest: ['background', 'purpose', 'joint', 'attribution']
  Furthest: ['concession', 'conclusion', 'condition', 'motivation']
Label: antithesis
  Closest: ['contrast', 'attribution', 'purpose', 'restatement']
  Furthest: ['comparison', 'solutionhood', 'preparation',
'evaluation']
Label: conclusion
  Closest: ['condition', 'motivation', 'concession', 'interpretation-
evaluation']
  Furthest: ['effect', 'background', 'joint', 'purpose']
Label: motivation
  Closest: ['condition', 'conclusion', 'concession', 'interpretation-
evaluation']
  Furthest: ['effect', 'background', 'joint', 'purpose']
```

Добавим в датасет столбцы с наиболее похожими и непохожими классами.

```
test_set["closest"] = test_set["label"].map(lambda x:
labels_dictionaty[x]["closest"])
test_set["furthest"] = test_set["label"].map(lambda x:
labels_dictionaty[x]["furthest"])
test_set["random"] = test_set["label"].map(
    lambda x: np.random.choice(
        test_set["label"].unique().tolist(), 4, replace=False
    ).tolist()
)
test_set["all"] = test_set["label"].map(lambda x:
test_set["label"].unique().tolist())
test_set.head()
```

```
                                                  start  \
6542           что если метафора на самом деле мертвая ,
21707                Полагаем , что первейшее условие ,
3770   вследствие чего он обладает средствами лингвис...
15858  « устраиваемая на звериных тропах большая запа...
22722       где я в крайнем случае и не помешаю никому ,

                                                    end
label  \
6542           она теряет свои композиционные свойства ,   condition

21707  В подавляющем большинстве недостатки произноше...       cause

3770                              является анализ того ,       joint
```

```
15858  - в значении « ловушка для рябчиков и куропато...  comparison

22722                                          если упаду ,   condition


                                                    closest  \
6542   [conclusion, motivation, concession, interpret...
21707          [cause-effect, joint, background, effect]
3770                [background, effect, cause, purpose]
15858  [solutionhood, preparation, evaluation, evidence]
22722  [conclusion, motivation, concession, interpret...

                                                   furthest  \
6542                [effect, background, joint, purpose]
21707  [restatement, interpretation-evaluation, motiv...
3770   [interpretation-evaluation, restatement, motiv...
15858       [antithesis, contrast, attribution, purpose]
22722               [effect, background, joint, purpose]

                                                    random  \
6542    [preparation, evaluation, cause-effect, purpose]
21707           [cause, attribution, purpose, condition]
3770   [background, conclusion, comparison, attribution]
15858        [effect, restatement, evidence, conclusion]
22722     [condition, effect, preparation, cause-effect]

                                                       all
6542   [condition, cause, joint, comparison, elaborat...
21707  [condition, cause, joint, comparison, elaborat...
3770   [condition, cause, joint, comparison, elaborat...
15858  [condition, cause, joint, comparison, elaborat...
22722  [condition, cause, joint, comparison, elaborat...
```

```python
label_to_emb = {row['label']: np.array(row['embedding']) for _, row in
labels_emb_df.iterrows()}

def avg_cosine_sim(col_name):
    sims = []
    for _, row in test_set.iterrows():
        label_emb = label_to_emb[row['label']]
        # Calculate average embedding for the choices
        choices_emb = np.mean([label_to_emb[choice] for choice in
row[col_name]], axis=0)

        sim = cosine_similarity([label_emb], [choices_emb])[0][0]
        sims.append(sim)
    return np.mean(sims)

similarities = {}
```

```python
for col in ["closest", "furthest", "random", "all"]:
    avg_sim = avg_cosine_sim(col)
    similarities[col] = avg_sim
    # print(f"Average cosine similarity for '{col}': {avg_sim:.4f}")
similarities_df = pd.DataFrame.from_dict(similarities, orient='index',
columns=['Average Cosine Similarity'])
similarities_df.head()
```

```
          Average Cosine Similarity
closest                    0.999531
furthest                   0.999327
random                     0.821814
all                        0.792919
```

Scoring

for this part to run you need to have `.env` file with `OPENAI_API_KEY` and `OPENAI_BASE_URL` set.

```python
from score_model import score_disrpt

# Посчитаем метрики для тестового набора с разными вариантами
вариантов ответа
closest_set = test_set[["start", "end", "label", "closest"]].copy()
closest_set.rename(columns={"closest": "choices"}, inplace=True)
score_disrpt(
    model_name="gpt-4o-mini",
    temperature=0.5,
    test_set=closest_set,
    filename="gpt4o_mini_closest.csv",
)
```

```
[INFO] Starting disrpt scoring...

Scoring disrpt: 100%|██████████| 2887/2887 [29:40<00:00,  1.62it/s]

[INFO] Disrpt scoring complete. Results saved to:
gpt4o_mini_closest.json



({'accuracy': 0.4624177346726706,
  'recall': 0.4987584860658019,
  'f1': 0.3712589679197093,
  'precision': 0.41347115981479604},
 'gpt4o_mini_closest.json')
```

```python
furthest_set = test_set[["start", "end", "label", "furthest"]].copy()
furthest_set.rename(columns={"furthest": "choices"}, inplace=True)
score_disrpt(
```

```
    model_name="gpt-4o-mini",
    temperature=0.5,
    test_set=furthest_set,
    filename="gpt4o_mini_furthest.csv",
)
```

```
[INFO] Starting disrpt scoring...

Scoring disrpt: 100%|██████████| 2887/2887 [30:16<00:00,  1.59it/s]

[INFO] Disrpt scoring complete. Results saved to:
gpt4o_mini_furthest.csv
```

```
({'accuracy': 0.4450987183927953,
  'recall': 0.4484420007167252,
  'f1': 0.3959464032324734,
  'precision': 0.4975411430200603},
 'gpt4o_mini_furthest.csv')
```

```
random_set = test_set[["start", "end", "label", "random"]].copy()
random_set.rename(columns={"random": "choices"}, inplace=True)
score_disrpt(
    model_name="gpt-4o-mini",
    temperature=0.5,
    test_set=random_set,
    filename="gpt4o_mini_random.csv",
)
```

```
[INFO] Starting disrpt scoring...

Scoring disrpt: 100%|██████████| 2887/2887 [29:41<00:00,  1.62it/s]

[INFO] Disrpt scoring complete. Results saved to:
gpt4o_mini_random.csv
```

```
({'accuracy': 0.4956702459300312,
  'recall': 0.5123448059758989,
  'f1': 0.35859194725027277,
  'precision': 0.3579938222156512},
 'gpt4o_mini_random.csv')
```

```
all_set = test_set[["start", "end", "label", "all"]].copy()
all_set.rename(columns={"all": "choices"}, inplace=True)
score_disrpt(
    model_name="gpt-4o-mini",
    temperature=0.5,
    test_set=all_set,
```

```
    filename="gpt4o_mini_all.csv",
)

[INFO] Starting disrpt scoring...

Scoring disrpt: 100%|██████████| 2887/2887 [36:08<00:00,  1.33it/s]

[INFO] Disrpt scoring complete. Results saved to: gpt4o_mini_all.json


/home/askatasuna/Документы/HSE/contexts/ContextBench_2025/.venv/lib/
python3.12/site-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/home/askatasuna/Документы/HSE/contexts/ContextBench_2025/.venv/lib/
python3.12/site-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

({'accuracy': 0.3349497748527884,
  'recall': 0.25376221773515817,
  'f1': 0.2031816823871577,
  'precision': 0.22613382686994632},
 'gpt4o_mini_all.json')
```

Analyzing results

Теперь мы можем проанализировать результаты, которые мы получили. В первую очередь, мы можем посмотреть на confusion matrix, чтобы понять, какие классы чаще всего путаются между собой. А также мы можем посмотреть на корреляцию accuracy и сабсета.

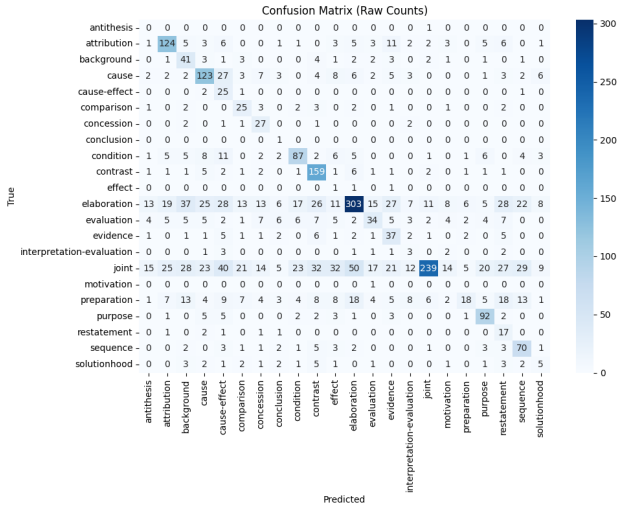
```
similar_labels_df = pd.read_csv("gpt4o_mini_closest.csv")
different_labels_df = pd.read_csv("gpt4o_mini_furthest.csv")
random_labels_df = pd.read_csv("gpt4o_mini_random.csv")
all_labels_df = pd.read_csv("gpt4o_mini_all.csv")
```

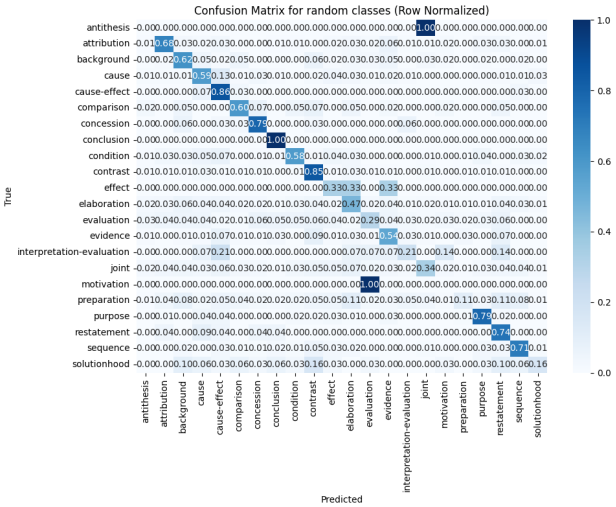
Случайный набор классов

```
y_true = random_labels_df["correct_answer"].values
y_pred = random_labels_df["answer"].values

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```python
# Get unique labels (relations)
labels = sorted(list(set(y_true) | set(y_pred)))

precision, recall, f1, _ = precision_recall_fscore_support(
    y_true, y_pred, average="macro", labels=labels
)
print(f"Macro Precision: {precision:.4f}")
print(f"Macro Recall: {recall:.4f}")
print(f"Macro F1: {f1:.4f}")

# Create figure with two subplots side by side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8))

# 1. Regular confusion matrix
cm = confusion_matrix(y_true, y_pred, labels=labels)
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=labels,
    yticklabels=labels,
    ax=ax1,
)
ax1.set_xlabel("Predicted")
ax1.set_ylabel("True")
ax1.set_title("Confusion Matrix (Raw Counts)")

# 2. Row-normalized confusion matrix (normalize by true label)
cm_normalized = confusion_matrix(y_true, y_pred, labels=labels,
normalize="true")
sns.heatmap(
    cm_normalized,
    annot=True,
    fmt=".2f",
    cmap="Blues",
    xticklabels=labels,
    yticklabels=labels,
    ax=ax2,
)
ax2.set_xlabel("Predicted")
ax2.set_ylabel("True")
ax2.set_title("Confusion Matrix for random classes (Row Normalized)")

plt.tight_layout()
plt.savefig("confusion_matrices.png", dpi=300)
plt.show()
```

```
Accuracy: 0.4957
Macro Precision: 0.3580
Macro Recall: 0.5123
Macro F1: 0.3586
```






```python
cm_df = pd.DataFrame(cm_normalized, index=labels, columns=labels)

misclassifications = []

for true_class in labels:
    for pred_class in labels:
        if true_class != pred_class:
            raw_count = cm[labels.index(true_class),
labels.index(pred_class)]

            # Only include pairs that actually appeared in the data
            if raw_count > 0:
                misclassifications.append(
                    {
                        "True Class": true_class,
                        "Predicted Class": pred_class,
                        "Error Rate": cm_df.loc[true_class,
pred_class],
                        "Count": raw_count,
                    }
                )

misclass_df = pd.DataFrame(misclassifications)
misclass_df = misclass_df.sort_values(by="Error Rate",
ascending=False)

print("Top confused pairs (by error rate):")
print(misclass_df.head(10))
```

```
misclass_df_by_count = misclass_df.sort_values(by="Count",
ascending=False)
print("\nTop confused pairs (by raw count):")
print(misclass_df_by_count.head(10))
```

```
Top confused pairs (by error rate):
                    True Class Predicted Class  Error Rate  Count
0                    antithesis           joint    1.000000      1
180                  motivation      evaluation    1.000000      1
96                       effect        evidence    0.333333      1
95                       effect     elaboration    0.333333      1
153  interpretation-evaluation    cause-effect    0.214286      3
237               solutionhood        contrast    0.161290      5
157  interpretation-evaluation      motivation    0.142857      2
158  interpretation-evaluation     restatement    0.142857      2
33                        cause    cause-effect    0.129187     27
199                preparation     restatement    0.108434     18

Top confused pairs (by raw count):
      True Class Predicted Class  Error Rate  Count
170        joint     elaboration    0.071327     50
163        joint    cause-effect    0.057061     40
99   elaboration      background    0.057099     37
168        joint        contrast    0.045649     32
169        joint          effect    0.045649     32
178        joint        sequence    0.041369     29
115  elaboration     restatement    0.043210     28
161        joint      background    0.039943     28
101  elaboration    cause-effect    0.043210     28
109  elaboration        evidence    0.041667     27
```

Весьма интересно, что accuracy на сабсете со случайными классами выше, чем на любом другом сабсете. Это может говорить о том, что модели могут не быть чувствительными к схожести классов, однако, это также может быть связано с тем, что при случайном выборе классов мы можем с небольшой вероятностью добавить целевой класс в варианты ответа дважды, что может повлиять на выбор модели.

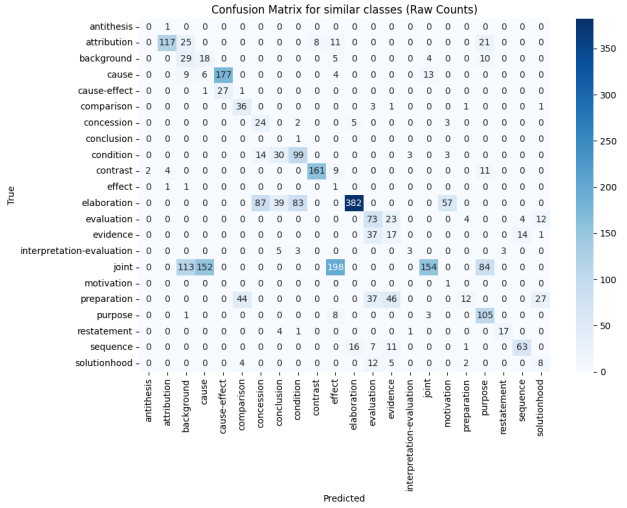
При этом мы можем заметить, что наибольшее смешение классов происходит на парах **joint-elaboration** и **joint-cause**.

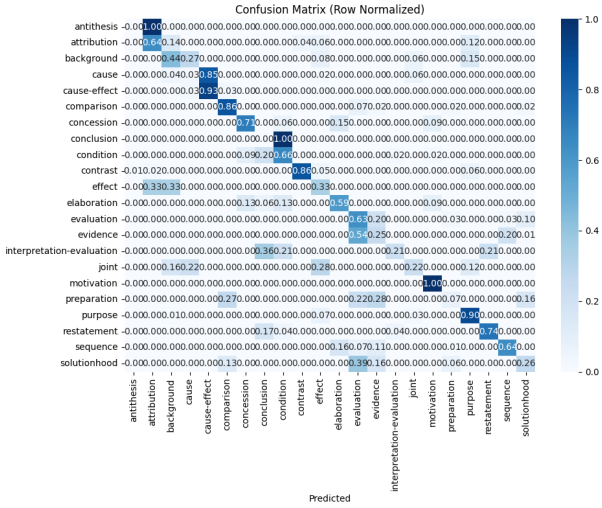
Наиболее похожие классы

```
y_true = similar_labels_df["correct_answer"].values
y_pred = similar_labels_df["answer"].values

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Get unique labels (relations)
```

```
labels = sorted(list(set(y_true) | set(y_pred)))

precision, recall, f1, _ = precision_recall_fscore_support(
    y_true, y_pred, average="macro", labels=labels
)
print(f"Macro Precision: {precision:.4f}")
print(f"Macro Recall: {recall:.4f}")
print(f"Macro F1: {f1:.4f}")

# Create figure with two subplots side by side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8))

# 1. Regular confusion matrix
cm = confusion_matrix(y_true, y_pred, labels=labels)
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=labels,
    yticklabels=labels,
    ax=ax1,
)
ax1.set_xlabel("Predicted")
ax1.set_ylabel("True")
ax1.set_title("Confusion Matrix for similar classes (Raw Counts)")

# 2. Row-normalized confusion matrix (normalize by true label)
cm_normalized = confusion_matrix(y_true, y_pred, labels=labels,
normalize="true")
sns.heatmap(
    cm_normalized,
    annot=True,
    fmt=".2f",
    cmap="Blues",
    xticklabels=labels,
    yticklabels=labels,
    ax=ax2,
)
ax2.set_xlabel("Predicted")
ax2.set_ylabel("True")
ax2.set_title("Confusion Matrix (Row Normalized)")

plt.tight_layout()
plt.savefig("confusion_matrices.png", dpi=300)
plt.show()

Accuracy: 0.4624
Macro Precision: 0.4135
```

```
Macro Recall: 0.4988
Macro F1: 0.3713
```






```
cm_df = pd.DataFrame(cm_normalized, index=labels, columns=labels)

misclassifications = []

for true_class in labels:
    for pred_class in labels:
        if true_class != pred_class:
            raw_count = cm[labels.index(true_class),
labels.index(pred_class)]

            # Only include pairs that actually appeared in the data
            if raw_count > 0:
                misclassifications.append(
                    {
                        "True Class": true_class,
                        "Predicted Class": pred_class,
                        "Error Rate": cm_df.loc[true_class,
pred_class],
                        "Count": raw_count,
                    }
                )

misclass_df = pd.DataFrame(misclassifications)
misclass_df = misclass_df.sort_values(by="Error Rate",
ascending=False)

print("Top confused pairs (by error rate):")
print(misclass_df.head(10))

misclass_df_by_count = misclass_df.sort_values(by="Count",
ascending=False)
```

```
print("\nTop confused pairs (by raw count):")
print(misclass_df_by_count.head(10))
```

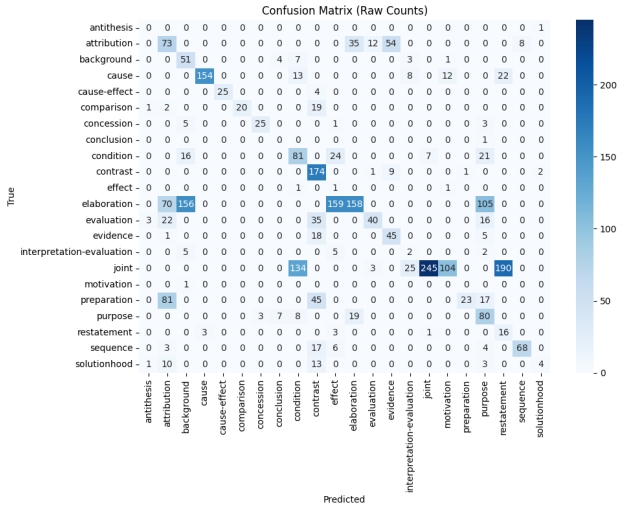
```
Top confused pairs (by error rate):
                   True Class Predicted Class  Error Rate  Count
0                   antithesis     attribution    1.000000      1
22                  conclusion       condition    1.000000      1
10                       cause    cause-effect    0.846890    177
41                    evidence      evaluation    0.536232     37
66                solutionhood      evaluation    0.387097     12
44  interpretation-evaluation      conclusion    0.357143      5
32                      effect      background    0.333333      1
31                      effect     attribution    0.333333      1
49                       joint          effect    0.282454    198
53                 preparation        evidence    0.277108     46

Top confused pairs (by raw count):
     True Class Predicted Class  Error Rate  Count
49        joint          effect    0.282454    198
10        cause    cause-effect    0.846890    177
48        joint           cause    0.216833    152
47        joint      background    0.161198    113
33  elaboration      concession    0.134259     87
50        joint         purpose    0.119829     84
35  elaboration       condition    0.128086     83
36  elaboration      motivation    0.087963     57
53  preparation        evidence    0.277108     46
51  preparation      comparison    0.265060     44
```

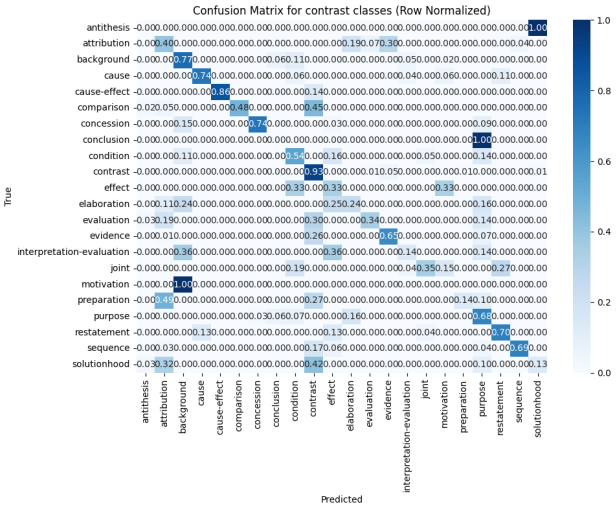
Различные классы

```
y_true = different_labels_df["correct_answer"].values
y_pred = different_labels_df["answer"].values

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Get unique labels (relations)
labels = sorted(list(set(y_true) | set(y_pred)))

precision, recall, f1, _ = precision_recall_fscore_support(
    y_true, y_pred, average="macro", labels=labels
)
print(f"Macro Precision: {precision:.4f}")
print(f"Macro Recall: {recall:.4f}")
print(f"Macro F1: {f1:.4f}")

# Create figure with two subplots side by side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8))
```

```
# 1. Regular confusion matrix
cm = confusion_matrix(y_true, y_pred, labels=labels)
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=labels,
    yticklabels=labels,
    ax=ax1,
)
ax1.set_xlabel("Predicted")
ax1.set_ylabel("True")
ax1.set_title("Confusion Matrix (Raw Counts)")

# 2. Row-normalized confusion matrix (normalize by true label)
cm_normalized = confusion_matrix(y_true, y_pred, labels=labels,
normalize="true")
sns.heatmap(
    cm_normalized,
    annot=True,
    fmt=".2f",
    cmap="Blues",
    xticklabels=labels,
    yticklabels=labels,
    ax=ax2,
)
ax2.set_xlabel("Predicted")
ax2.set_ylabel("True")
ax2.set_title("Confusion Matrix for contrast classes (Row
Normalized)")

plt.tight_layout()
plt.savefig("confusion_matrices.png", dpi=300)
plt.show()
```

```
Accuracy: 0.4451
Macro Precision: 0.4975
Macro Recall: 0.4484
Macro F1: 0.3959
```






```
cm_df = pd.DataFrame(cm_normalized, index=labels, columns=labels)

misclassifications = []

for true_class in labels:
    for pred_class in labels:
        if true_class != pred_class:
            raw_count = cm[labels.index(true_class),
labels.index(pred_class)]

            # Only include pairs that actually appeared in the data
            if raw_count > 0:
                misclassifications.append(
                    {
                        "True Class": true_class,
                        "Predicted Class": pred_class,
                        "Error Rate": cm_df.loc[true_class,
pred_class],
                        "Count": raw_count,
                    }
                )

misclass_df = pd.DataFrame(misclassifications)
misclass_df = misclass_df.sort_values(by="Error Rate",
ascending=False)

print("Top confused pairs (by error rate):")
print(misclass_df.head(10))

misclass_df_by_count = misclass_df.sort_values(by="Count",
ascending=False)
print("\nTop confused pairs (by raw count):")
print(misclass_df_by_count.head(10))
```

```
Top confused pairs (by error rate):
                   True Class Predicted Class  Error Rate  Count
0                   antithesis    solutionhood    1.000000      1
50                  motivation      background    1.000000      1
20                  conclusion         purpose    1.000000      1
51                 preparation     attribution    0.487952     81
16                  comparison        contrast    0.452381     19
67                solutionhood        contrast    0.419355     13
43  interpretation-evaluation          effect    0.357143      5
42  interpretation-evaluation      background    0.357143      5
29                     effect       condition    0.333333      1
30                     effect      motivation    0.333333      1

Top confused pairs (by raw count):
     True Class Predicted Class  Error Rate  Count
49        joint     restatement    0.271041    190
33  elaboration          effect    0.245370    159
32  elaboration      background    0.240741    156
45        joint       condition    0.191155    134
34  elaboration         purpose    0.162037    105
48        joint      motivation    0.148359    104
51  preparation     attribution    0.487952     81
31  elaboration     attribution    0.108025     70
3   attribution        evidence    0.296703     54
52  preparation        contrast    0.271084     45
```

Все классы

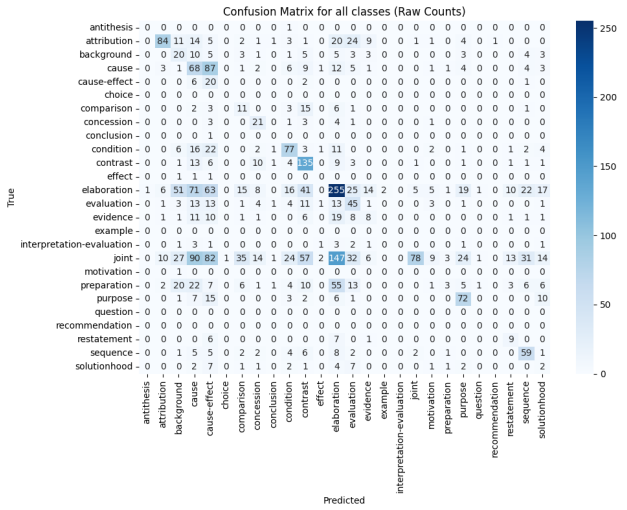
```python
y_true = all_labels_df["correct_answer"].values
y_pred = all_labels_df["answer"].values

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Get unique labels (relations)
labels = sorted(list(set(y_true) | set(y_pred)))

precision, recall, f1, _ = precision_recall_fscore_support(
    y_true, y_pred, average="macro", labels=labels
)
print(f"Macro Precision: {precision:.4f}")
print(f"Macro Recall: {recall:.4f}")
print(f"Macro F1: {f1:.4f}")

# Create figure with two subplots side by side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8))

# 1. Regular confusion matrix
cm = confusion_matrix(y_true, y_pred, labels=labels)
```

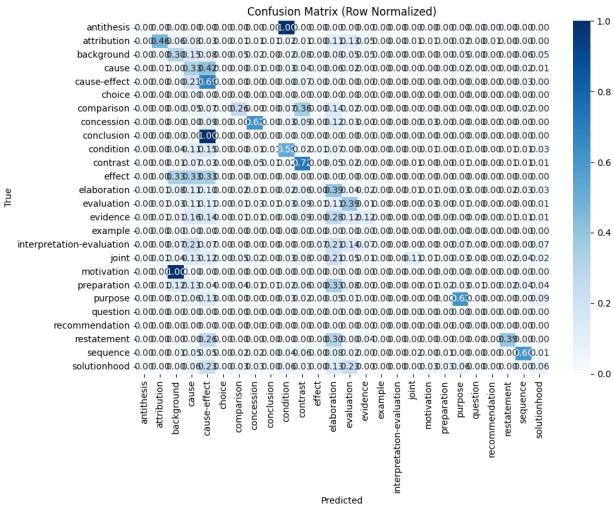
```
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=labels,
    yticklabels=labels,
    ax=ax1,
)
ax1.set_xlabel("Predicted")
ax1.set_ylabel("True")
ax1.set_title("Confusion Matrix for all classes (Raw Counts)")

# 2. Row-normalized confusion matrix (normalize by true label)
cm_normalized = confusion_matrix(y_true, y_pred, labels=labels,
normalize="true")
sns.heatmap(
    cm_normalized,
    annot=True,
    fmt=".2f",
    cmap="Blues",
    xticklabels=labels,
    yticklabels=labels,
    ax=ax2,
)
ax2.set_xlabel("Predicted")
ax2.set_ylabel("True")
ax2.set_title("Confusion Matrix (Row Normalized)")

plt.tight_layout()
plt.savefig("confusion_matrices.png", dpi=300)
plt.show()
```

```
Accuracy: 0.3349
Macro Precision: 0.2261
Macro Recall: 0.2538
Macro F1: 0.2032
```

```
/home/askatasuna/Документы/HSE/contexts/ContextBench_2025/.venv/lib/
python3.12/site-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/home/askatasuna/Документы/HSE/contexts/ContextBench_2025/.venv/lib/
python3.12/site-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```





Поиск корреляций

task	precision
contrasting	0.4975
similar	0.4135
random	0.3580
all	0.2261

Итак, мы увидели, что precision в целом согласуется с нашими ожиданиями: наибольшее значение precision у сабсета с наиболее различными классами, что говорит о том, что модель с бОльшим успехом различает классы менее похожи по косинусной близости. Проверим эти данных также с помощью корреляции precision и среднего косинусного расстояния между лейблом и вариантами ответа в сабсете.

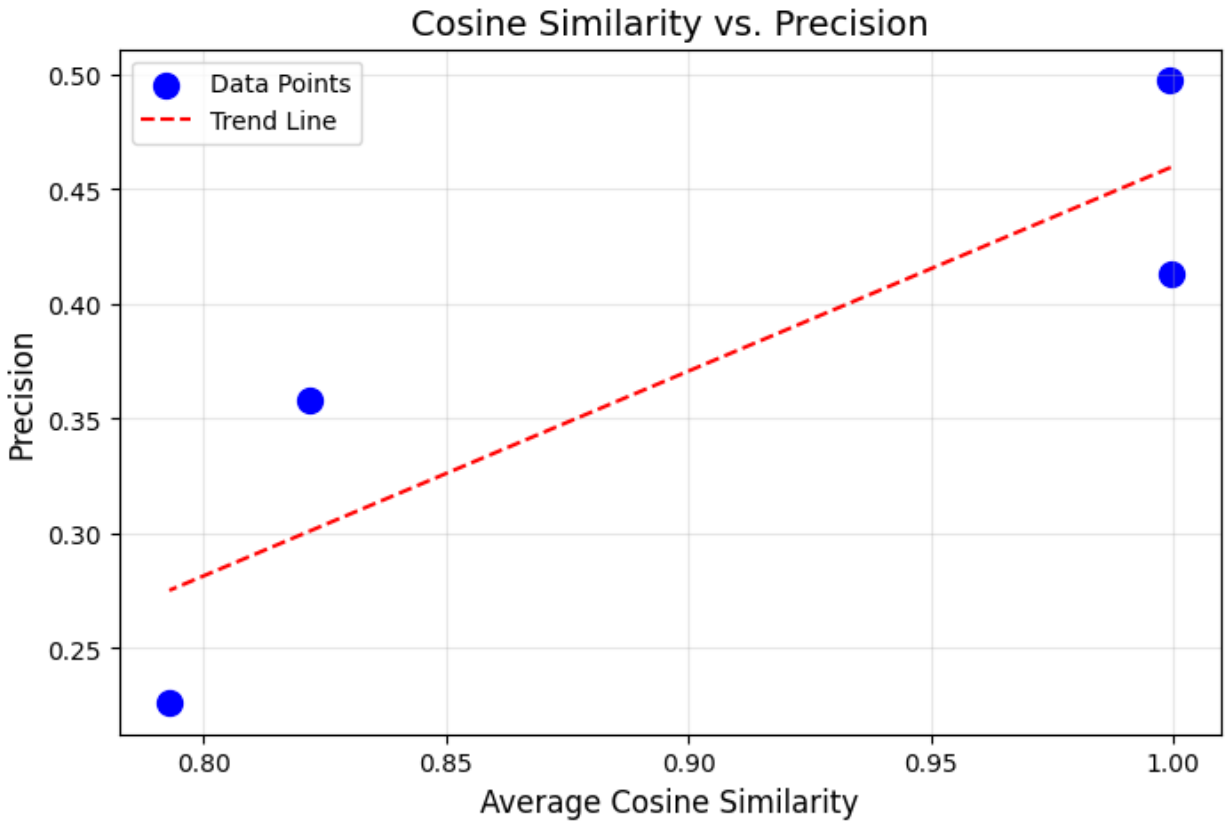
```
similarities_df.head()
```

```
          Average Cosine Similarity  Precision
closest                    0.999531     0.4135
furthest                   0.999327     0.4975
random                     0.821814     0.3580
all                        0.792919     0.2261
```

```
similarities_df['Precision'] = [0.4135, 0.4975, 0.3580, 0.2261]
similarities_df.head()
```

```
          Average Cosine Similarity  Precision
closest                    0.999531     0.4135
furthest                   0.999327     0.4975
random                     0.821814     0.3580
all                        0.792919     0.2261
```

```
plt.figure(figsize=(8, 5))
plt.scatter(
    similarities_df['Average Cosine Similarity'],
    similarities_df['Precision'],
    color='blue',
    s=100,
    label='Data Points'
)

plt.xlabel('Average Cosine Similarity', fontsize=12)
plt.ylabel('Precision', fontsize=12)
plt.title('Cosine Similarity vs. Precision', fontsize=14)

# for i, row in similarities_df.iterrows():
#     plt.annotate(row['Group'], (row['Average Cosine Similarity'],
row['Precision']),
#                  textcoords="offset points", xytext=(5,5),
ha='left')

import numpy as np
x = similarities_df['Average Cosine Similarity']
y = similarities_df['Precision']
m, b = np.polyfit(x, y, 1)
plt.plot(x, m*x + b, color='red', linestyle='--', label='Trend Line')

plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```




```
corr, p_value = pearsonr(similarities_df['Average Cosine Similarity'],
similarities_df['Precision'])
print(f"Pearson correlation: {corr:.3f}, p-value: {p_value:.4f}")

Pearson correlation: 0.873, p-value: 0.1266
```

Итак, корреляция будто бы и есть, хотя и противоречащая нашей гипотезе H0, но с низким p-value, что говорит о том, что мы не можем отвергнуть нулевую гипотезу. Вытащить из этого какие-то более глубокие выводы не представляется возможным, т.к. у нас нет достаточного количества данных для статистически значимого вывода.

Выводы

Мы не получили надежных данных о том, что семантическая близость классов влияет на precision LLM классификатора, т.к. корреляция между precision и семантической близостью классов не является статистически значимой. Это может быть следствием того, что ruBERT не справляется с достаточно сильным разеделением классов в датасете (используя только названия классов). Об этом также может свидетельствовать высокий precision на сабсете со случайными классами и достаточно большое смешение классов в confusion matrix в остальных случаях.