fitz

May 5, 2024

1 Github repo: https://github.com/lthroy/CS598DLHFinal

2 Video presentation: https://mediaspace.illinois.edu/media/t/1_srkvifnx

[1]:
```
from src import platt
from src import uncertainty
from src import conformal
from src import distances
from src import metrics
from src import utils

import importlib
importlib.reload(metrics)
importlib.reload(platt)
importlib.reload(uncertainty)
importlib.reload(conformal)
importlib.reload(utils)
importlib.reload(distances)

import collections, itertools, json, os, pathlib, sys

import PIL
import scipy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt; plt.style.use('bmh')
import matplotlib.pyplot as plt
plt.rcParams['pdf.fonttype'] = 42
plt.rcParams['ps.fonttype'] = 42
plt.rcParams["font.family"] = "Times New Roman"

import torch
print('scipy\t', scipy.__version__)
print('pytorch\t', torch.__version__)
print('pandas\t', pd.__version__)
print('numpy\t', np.__version__)
print('python\t', sys.version)
```

```
#print('matplotlib\t',matplotlib.__version__)
np.random.seed(0)
import time
start_time = time.time()

RUNS = 5
```

```
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages/tqdm/auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
scipy    1.7.3
pytorch  1.10.2
pandas   1.3.4
numpy    1.22.2
python   3.8.18 | packaged by conda-forge | (default, Dec 23 2023, 17:25:47)
[Clang 16.0.6 ]
```

```
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
packages/torchvision/io/image.py:11: UserWarning: Failed to load image Python
extension: dlopen(/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
packages/torchvision/image.so, 0x0006): Library not loaded:
@rpath/libpng16.16.dylib
  Referenced from: <8E4D9E61-A0A3-30A7-B778-23E3E702B690>
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages/torchvision/image.so
  Reason: tried: '/opt/anaconda3/envs/dlh38env/lib/python3.8/lib-
dynload/../../libpng16.16.dylib' (no such file),
'/opt/anaconda3/envs/dlh38env/bin/../lib/libpng16.16.dylib' (no such file),
'/usr/local/lib/libpng16.16.dylib' (no such file), '/usr/lib/libpng16.16.dylib'
(no such file, not in dyld cache)
  warn(f"Failed to load image Python extension: {e}")
```

3 Introduction

Deep learning has the potential to automate many clinically useful tasks in medical imaging. However the translation of deep learning into clinical practice has been hindered by issues such as lack of the transparency and interpretability in these "black box" algorithms compared to traditional statistical methods. Specifically, many clinical deep learning models lack rigorous and robust techniques for conveying certainty (or lack thereof) in their predictions – ultimately limiting their appeal for extensive use in medical decision-making. Furthermore, numerous demonstrations of algorithmic bias have increased hesitancy towards deployment of deep learning for clinical applications.

To this end, the authors explore how conformal predictions can complement existing deep learning approaches by providing an intuitive way of expressing uncertainty while facilitating greater transparency to clinical users. In this paper, the authors conduct field interviews with radiologists to assess possible use-cases for conformal predictors. Using insights gathered from these interviews, the authors devise two clinical use-cases and empirically evaluate several methods of conformal predictions on a dermatology photography dataset for skin lesion classification. The authors show how

to modify conformal predictions to be more adaptive to subgroup differences in patient skin tones through equalized coverage. Finally, the authors compare conformal prediction against measures of epistemic uncertainty.

The authors found all four methods perform similarly at a specific miscoverage parameter in terms of acuracy and set size at subgroup level. They observed lower subgroup coverage as well as higher set size disparity with group varients than standard aggregate conformal methods. At last, they found a strong correlation between set size and epistemic uncertainty. The paper suggested several ways in which conformal predictions could allow for greater transparency in medical AI, and formulated two general use-cases for conformal predictions for clincical decision making.

4 Scope of Reproducibility:

List hypotheses from the paper you will test and the corresponding experiments you will run.

1. Compare the following conformal prediction methods

   i. Naive

   ii. Adaptive prediction set (APS)

   iii. Groups APS (GAPS)

   iv. Regularized adaptive prediction set (RAPS)

   v. Group RAPS (GRAPS)

2. We will compare the following aspects

   i. Coverage

   ii. Cardinality

   iii. Distribution of prediction set sizes by subgroup

   iv. Rule-in v.s. rule-out scenerios

   v. Comparing conformal uncertainty to epistemic uncertainty

5 Methodology

This methodology is the core of your project. It consists of run-able codes with necessary annotations to show the expeiment you executed for testing the hypotheses.

The methodology at least contains two subsections **data** and **model** in your experiment.

5.1 Envioronment

5.1.1 Python version

3.8 ### Download from Github

```
git clone https://github.com/lthroy/CS598DLHFinal.git
```

5.1.2 Dependencies/packages needed

In command line, run the following command to install the necessary dependencies

```
pip install -r requirements.txt
```

5.2 Data

5.2.1 Data download instruction

Download the source image data from Google Drive and unzip

https://drive.google.com/file/d/18wvJGDnAlhSRov3Z2ShVwjPy-_EGN22Q/view?usp=drive_link

Download the serialized training results from Google Drive

https://drive.google.com/drive/folders/1paF6XRpFotOmCrJ4izckNVXGl9q93bRI?usp=drive_link

If the folders are too large, you can just download test_dfs.pickle and valid_dfs.pickle

Your folder structure should look like

```
CS598DLHFinal/

  README.md

  src/


  fritz17k/


  run_result/
|   |  valid_dfs.pickle
|     test_dfs.pickle
|   |--- ...
|
|
|
  requirements.txt

  skin_info2.csv

  fitz.ipynb
```

5.2.2 Source of the data

This paper uses the Fitzpatrick 17k dataset, which is available at https://github.com/mattgroh/fitzpatrick17k. This dataset contains roughly 17k clincal images labeled with skin conditions and Fitzpatrick skin types. Out of which 12672 images came from DermaAmin and 3,905 images came from Atlas Dermatologico.

5.2.3 Data descriptions with helpful charts and visualizations

[2]:
```
label_csv = 'skin_info2.csv'
skin_df = pd.read_csv(label_csv)
skin_df.head()
```

[2]:

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	md5hash	fitzpatrick	raw_label	nine_partition_label	three_partition_label	qc	url	url_alphanum
0	0	0	0	5e82a45bc5d78bd24ae9202d194423f8	3	drug induced pigmentary changes	inflammatory	non-neoplastic	NaN	https://www.dermaamin.com/site/images/clinical…	httpwwwdermaamincomsiteimagesclinicalpicmminoc…
1	1	1	1	fa2911a9b13b6f8af79cb700937cc14f	1	photodermatoses	inflammatory	non-neoplastic	NaN	https://www.dermaamin.com/site/images/clinical…	httpwwwdermaamincomsiteimagesclinicalpicpphoto…
2	2	2	2	d2bac3c9e4499032ca8e9b07c7d3bc40	2	dermatofibroma	benign dermal	benign	NaN	https://www.dermaamin.com/site/images/clinical…	httpwwwdermaamincomsiteimagesclinicalpicdderma…
3	3	3	3	0a94359e7eaacd7178e06b2823777789	1	psoriasis	inflammatory	non-neoplastic	NaN	https://www.dermaamin.com/site/images/clinical…	httpwwwdermaamincomsiteimagesclinicalpicppsori…
4	4	4	4	a39ec3b1f22c08a421fa20535e037bba	1	psoriasis	inflammatory	non-neoplastic	NaN	https://www.dermaamin.com/site/images/clinical…	httpwwwdermaamincomsiteimagesclinicalpicppsori…

```
                                  image  label  source
0  5e82a45bc5d78bd24ae9202d194423f8.jpg      4       0
1  fa2911a9b13b6f8af79cb700937cc14f.jpg      4       0
2  d2bac3c9e4499032ca8e9b07c7d3bc40.jpg      0       0
3  0a94359e7eaacd7178e06b2823777789.jpg      4       0
4  a39ec3b1f22c08a421fa20535e037bba.jpg      4       0
```

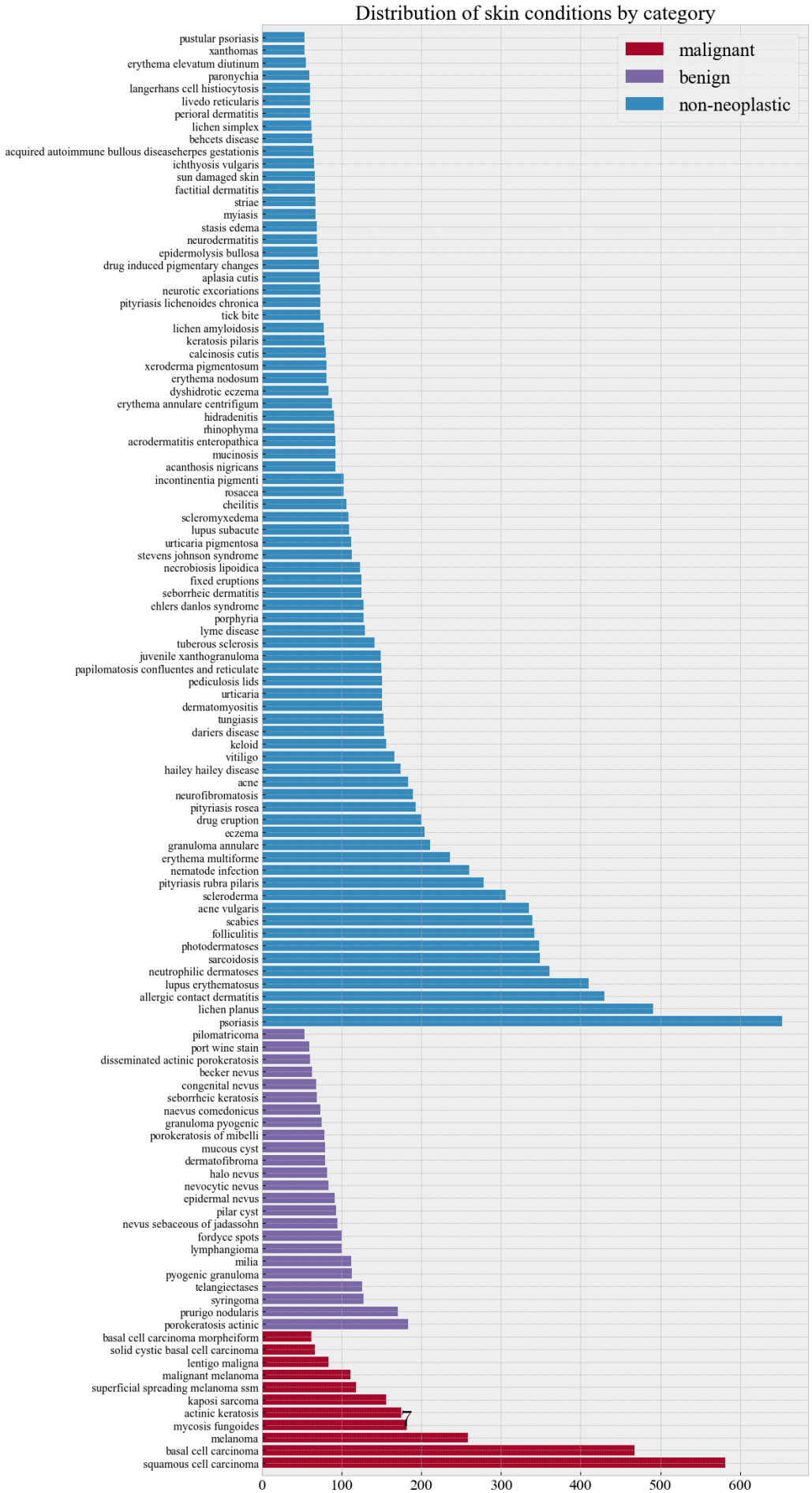
[3]:
```
save_dir = 'figures/'
fig_dir = pathlib.Path(save_dir)
fig_dir.mkdir(exist_ok=True)
```

[4]:
```
neo_df = skin_df.query('three_partition_label == "non-neoplastic"')
ben_df = skin_df.query('three_partition_label == "benign"')
mal_df = skin_df.query('three_partition_label == "malignant"')
```

Class distributions There are total 114 classes:

- They can be further categorized as malignant,benign and non-neoplastic

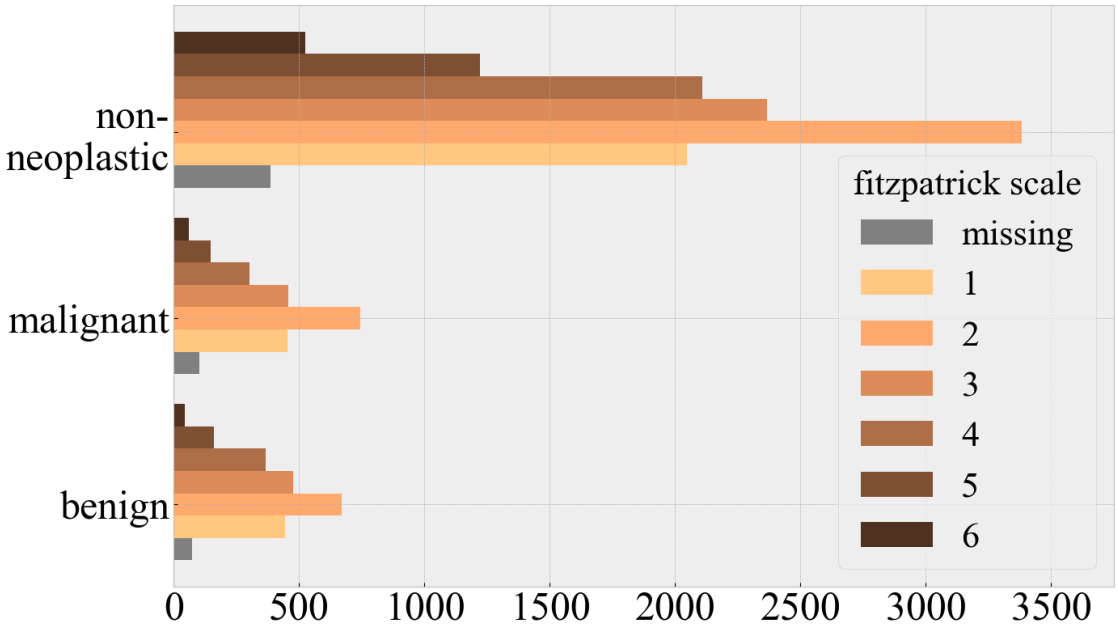
[5]:
```
plt.figure(figsize=(12, 22))
plt.title('Distribution of skin conditions by category', fontsize=22)
plt.ylim(-1, 114)
plt.xticks(fontsize=16)
plt.yticks(fontsize=12)
# plt.ylabel('Skin condition', fontsize=24)
plt.barh(*list(zip(*mal_df.raw_label.value_counts().items())), color='C1',␣
 ↪label='malignant')
plt.barh(*list(zip(*ben_df.raw_label.value_counts().items())), color='C2',␣
 ↪label='benign')
plt.barh(*list(zip(*neo_df.raw_label.value_counts().items())), color='C0',␣
 ↪label='non-neoplastic')
plt.legend(fontsize=20)
plt.tight_layout()
plt.savefig(fig_dir / 'bar-class-distribution.png')
plt.show()
```


Distribution of skin conditions by category
malignant
benign
non-neoplastic
pustular psoriasis
xanthomas
erythema elevatum diutinum
paronychia
langerhans cell histiocytosis
livedo reticularis
perioral dermatitis
lichen simplex
behcets disease
acquired autoimmune bullous diseaseherpes gestationis
ichthyosis vulgaris
sun damaged skin
factitial dermatitis
striae
myiasis
stasis edema
neurodermatitis
epidermolysis bullosa
drug induced pigmentary changes
aplasia cutis
neurotic excoriations
pityriasis lichenoides chronica
tick bite
lichen amyloidosis
keratosis pilaris
calcinosis cutis
xeroderma pigmentosum
erythema nodosum
dyshidrotic eczema
erythema annulare centrifigum
hidradenitis
rhinophyma
acrodermatitis enteropathica
mucinosis
acanthosis nigricans
incontinentia pigmenti
rosacea
cheilitis
scleromyxedema
lupus subacute
urticaria pigmentosa
stevens johnson syndrome
necrobiosis lipoidica
fixed eruptions
seborrheic dermatitis
ehlers danlos syndrome
porphyria
lyme disease
tuberous sclerosis
juvenile xanthogranuloma
papilomatosis confluentes and reticulate
pediculosis lids
urticaria
dermatomyositis
tungiasis
dariers disease
keloid
vitiligo
hailey hailey disease
acne
neurofibromatosis
pityriasis rosea
drug eruption
eczema
granuloma annulare
erythema multiforme
nematode infection
pityriasis rubra pilaris
scleroderma
acne vulgaris
scabies
folliculitis
photodermatoses
sarcoidosis
neutrophilic dermatoses
lupus erythematosus
allergic contact dermatitis
lichen planus
psoriasis
pilomatricoma
port wine stain
disseminated actinic porokeratosis
becker nevus
congenital nevus
seborrheic keratosis
naevus comedonicus
granuloma pyogenic
porokeratosis of mibelli
mucous cyst
dermatofibroma
halo nevus
nevocytic nevus
epidermal nevus
pilar cyst
nevus sebaceous of jadassohn
fordyce spots
lymphangioma
milia
pyogenic granuloma
telangiectases
syringoma
prurigo nodularis
porokeratosis actinic
basal cell carcinoma morpheiform
solid cystic basal cell carcinoma
lentigo maligna
malignant melanoma
superficial spreading melanoma ssm
kaposi sarcoma
actinic keratosis
mycosis fungoides
melanoma
basal cell carcinoma
squamous cell carcinoma
0
100
200
300
400
500
600

The distribution of malignant,benign and non-neoplastic by fitzpatrick scale (skin color)

[6]:
```
import matplotlib.pyplot as plt
import numpy as np

SKIN_COLORS = plt.cm.copper_r(np.linspace(0, 0.75, 6))
fontsize=24

classes = ['benign', 'malignant', 'non-\nneoplastic']

x = np.arange(len(classes))  # the label locations
width = 0.12  # the width of the bars

fig, ax = plt.subplots(figsize=(14, 8))

bars = [
    ax.barh(
        x + (-2 if i == -1 else i - 2) * width,
        list(map(
            lambda x: x[1],
            sorted(skin_df.query(f'fitzpatrick == @i').three_partition_label.
 ↪value_counts(sort=False).items())
        )),
        width,
        label='missing' if i == -1 else str(i),
        color='gray' if i == -1 else SKIN_COLORS[i-1],
    )
    for i in [-1] + list(range(1, 7))

]

# ax.set_title('Distribution of disease type by skin tone', fontsize=fontsize +␣
 ↪16)
ax.set_xlim(0, 3750)
ax.set_xticks([0, 500, 1000, 1500, 2000, 2500, 3000, 3500])
for t in ax.get_xticklabels():    #get_xticklabels will get you the label␣
 ↪objects, same for y
    t.set_fontsize(fontsize + 12)
ax.set_yticks(x, classes, fontsize=fontsize + 12)
# ax.set_ylabel('Skin condition type', fontsize=fontsize + 8)

ax.legend(
    title='fitzpatrick scale',
    title_fontproperties={
```

```
        'style': 'normal',
        'size': fontsize + 8,
    },
    fontsize=fontsize + 8,
    loc='lower right',
)

# for bar in bars:
    # ax.bar_label(bar, padding=4, fontsize=fontsize+6)

fig.tight_layout()
plt.savefig(fig_dir / 'fitz-subgroup-class-dist.pdf')
plt.show()
```



Disease prevalence by skin type

[7]:
```
skin_type_1 = skin_df.query('fitzpatrick == 1').three_partition_label.
 ↪value_counts()
skin_type_2 = skin_df.query('fitzpatrick == 2').three_partition_label.
 ↪value_counts()
skin_type_3 = skin_df.query('fitzpatrick == 3').three_partition_label.
 ↪value_counts()
skin_type_4 = skin_df.query('fitzpatrick == 4').three_partition_label.
 ↪value_counts()
skin_type_5 = skin_df.query('fitzpatrick == 5').three_partition_label.
 ↪value_counts()
```

```
skin_type_6 = skin_df.query('fitzpatrick == 6').three_partition_label.
 ↪value_counts()
skin_type_missing = skin_df.query('fitzpatrick == -1').three_partition_label.
 ↪value_counts()
skin_type_all = skin_df.three_partition_label.value_counts()

def prevalence(group):
    total = group.sum()
    neo = group['non-neoplastic'] / total
    ben = group['benign'] / total
    mal = group['malignant'] / total
    print(f'Non-neoplastic\t{neo:.2%}')
    print(f'Benign\t\t{ben:.2%}')
    print(f'Malignant\t{mal:.2%}')
    print(f'Count\t\t{total:.2f}')
    print()

print('1'.center(20, '-')); prevalence(skin_type_1)
print('2'.center(20, '-')); prevalence(skin_type_2)
print('3'.center(20, '-')); prevalence(skin_type_3)
print('4'.center(20, '-')); prevalence(skin_type_4)
print('5'.center(20, '-')); prevalence(skin_type_5)
print('6'.center(20, '-')); prevalence(skin_type_6)
print('Missing'.center(20, '-')); prevalence(skin_type_missing)
print('All'.center(20, '-')); prevalence(skin_type_all)
```

```
---------1----------
Non-neoplastic  69.57%
Benign          15.03%
Malignant       15.40%
Count           2941.00

---------2----------
Non-neoplastic  70.54%
Benign          13.99%
Malignant       15.47%
Count           4796.00

---------3----------
Non-neoplastic  71.76%
Benign          14.41%
Malignant       13.83%
Count           3297.00

---------4----------
Non-neoplastic  75.93%
Benign          13.23%
```

```
Malignant       10.85%
Count           2775.00

---------5----------
Non-neoplastic  80.03%
Benign          10.41%
Malignant       9.56%
Count           1527.00

---------6----------
Non-neoplastic  83.44%
Benign          7.01%
Malignant       9.55%
Count           628.00

------Missing-------
Non-neoplastic  68.86%
Benign          12.99%
Malignant       18.15%
Count           562.00

--------All---------
Non-neoplastic  72.82%
Benign          13.50%
Malignant       13.68%
Count           16526.00
```

5.2.4 Preprocessing code + command

Some random flip, random rotation, random resized crop and random affine transformations and more are implemented in ./src/dataset.py. It is embedded into the training process.

5.3 Model

5.3.1 Citation to the original paper

```
@article{Lu_Lemay_Chang_Höbel_Kalpathy-Cramer_2022,
title={Fair Conformal Predictors for Applications in Medical Imaging},
volume={36},
url={https://ojs.aaai.org/index.php/AAAI/article/view/21459},
DOI={10.1609/aaai.v36i11.21459},
number={11},
journal={Proceedings of the AAAI Conference on Artificial Intelligence},
author={Lu, Charles and Lemay, Andréanne and Chang, Ken and Höbel, Katharina and Kalpathy-Crame
year={2022},
month={Jun.},
pages={12008-12016}
}
```

5.3.2 Link to the original paper's repo

https://github.com/clu5/AAAI-22/tree/main

5.3.3 Model description

In the paper, the author directly used pretained ResNet-18 model trained with ImageNet data that is available for download from PyTorch.

5.3.4 Implementation code

In the `./src` folder, `resnet.py` contains the code to fetch different flavors of the ResNet models like ResNet-18 and ResNet-34. `dropout_resnet.py` contains an experiment of the ResNet model without the dropout layers that wasn't used in the final implementation. Run `train.py` to train the model with different parameters.

5.3.5 Pretrained model

The authors directly used a pretrained version of the ResNet-18 model. See `resnet.py` under `model_urls` for the download urls.

5.4 Training

5.4.1 Hyperparameters Used

- batch_size:16
- learning_rate: 1e-4
- dropout_rate: 0.5

5.4.2 Computational requirement:

- Hardware: Mac Mini (2020, M1 Chip, 8GB RAM)
- Time: Each seed (with early stopping set to 5, runs around 30 epochs per seed) takes around 6 hours, averaing 12 minutes per epoch; in the end of training there are 30 Monte-Carlo dropouts for inference (this is required because we want to quantify uncertainty)
- Seeds (different initiations): 5
- Total hours used: Around 30 hours in total
- Training epochs: 100, with early stopping = 5. The loss plateaus around 30 epochs per seed

5.4.3 Training code

The following is a test run only.

In reality, we should run something like the following in a command line interface

```
python3 src/train.py --runs 0 --epochs 100 --arch resnet18 --early_stop 5
python3 src/train.py --runs 1 --epochs 100 --arch resnet18 --early_stop 5
python3 src/train.py --runs 2 --epochs 100 --arch resnet18 --early_stop 5
python3 src/train.py --runs 3 --epochs 100 --arch resnet18 --early_stop 5
python3 src/train.py --runs 4 --epochs 100 --arch resnet18 --early_stop 5
```

The "runs" parameter is the seed I mentioned above, we want to run the model with 5 different initial parameter values (seeds), to save time, I have put the results in the folder "run_result" which you should have downloaded

[8]:
```
# A test run of the training script in Debug mode: faster
!python3 src/train.py --debug --runs 0 --epochs 2 --arch resnet18 --early_stop␣
 ↪1 --monte_carlo 3
```

```
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
packages/torchvision/io/image.py:11: UserWarning: Failed to load image Python
extension: dlopen(/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
packages/torchvision/image.so, 0x0006): Library not loaded:
@rpath/libpng16.16.dylib
  Referenced from: <8E4D9E61-A0A3-30A7-B778-23E3E702B690>
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages/torchvision/image.so
  Reason: tried: '/opt/anaconda3/envs/dlh38env/lib/python3.8/lib-
dynload/../../libpng16.16.dylib' (no such file),
'/opt/anaconda3/envs/dlh38env/bin/../lib/libpng16.16.dylib' (no such file),
'/usr/local/lib/libpng16.16.dylib' (no such file), '/usr/lib/libpng16.16.dylib'
(no such file, not in dyld cache)
  warn(f"Failed to load image Python extension: {e}")
--- class_map: {'acanthosis nigricans': 0, 'acne': 1, 'acne vulgaris': 2,
'acquired autoimmune bullous diseaseherpes gestationis': 3, 'acrodermatitis
enteropathica': 4, 'actinic keratosis': 5, 'allergic contact dermatitis': 6,
'aplasia cutis': 7, 'basal cell carcinoma': 8, 'basal cell carcinoma
morpheiform': 9, 'becker nevus': 10, 'behcets disease': 11, 'calcinosis cutis':
12, 'cheilitis': 13, 'congenital nevus': 14, 'dariers disease': 15,
'dermatofibroma': 16, 'dermatomyositis': 17, 'disseminated actinic
porokeratosis': 18, 'drug eruption': 19, 'drug induced pigmentary changes': 20,
'dyshidrotic eczema': 21, 'eczema': 22, 'ehlers danlos syndrome': 23, 'epidermal
nevus': 24, 'epidermolysis bullosa': 25, 'erythema annulare centrifigum': 26,
'erythema elevatum diutinum': 27, 'erythema multiforme': 28, 'erythema nodosum':
29, 'factitial dermatitis': 30, 'fixed eruptions': 31, 'folliculitis': 32,
'fordyce spots': 33, 'granuloma annulare': 34, 'granuloma pyogenic': 35, 'hailey
hailey disease': 36, 'halo nevus': 37, 'hidradenitis': 38, 'ichthyosis
vulgaris': 39, 'incontinentia pigmenti': 40, 'juvenile xanthogranuloma': 41,
'kaposi sarcoma': 42, 'keloid': 43, 'keratosis pilaris': 44, 'langerhans cell
histiocytosis': 45, 'lentigo maligna': 46, 'lichen amyloidosis': 47, 'lichen
planus': 48, 'lichen simplex': 49, 'livedo reticularis': 50, 'lupus
erythematosus': 51, 'lupus subacute': 52, 'lyme disease': 53, 'lymphangioma':
54, 'malignant melanoma': 55, 'melanoma': 56, 'milia': 57, 'mucinosis': 58,
'mucous cyst': 59, 'mycosis fungoides': 60, 'myiasis': 61, 'naevus comedonicus':
62, 'necrobiosis lipoidica': 63, 'nematode infection': 64, 'neurodermatitis':
65, 'neurofibromatosis': 66, 'neurotic excoriations': 67, 'neutrophilic
dermatoses': 68, 'nevocytic nevus': 69, 'nevus sebaceous of jadassohn': 70,
'papilomatosis confluentes and reticulate': 71, 'paronychia': 72, 'pediculosis
lids': 73, 'perioral dermatitis': 74, 'photodermatoses': 75, 'pilar cyst': 76,
'pilomatricoma': 77, 'pityriasis lichenoides chronica': 78, 'pityriasis rosea':
```

```
79, 'pityriasis rubra pilaris': 80, 'porokeratosis actinic': 81, 'porokeratosis
of mibelli': 82, 'porphyria': 83, 'port wine stain': 84, 'prurigo nodularis':
85, 'psoriasis': 86, 'pustular psoriasis': 87, 'pyogenic granuloma': 88,
'rhinophyma': 89, 'rosacea': 90, 'sarcoidosis': 91, 'scabies': 92,
'scleroderma': 93, 'scleromyxedema': 94, 'seborrheic dermatitis': 95,
'seborrheic keratosis': 96, 'solid cystic basal cell carcinoma': 97, 'squamous
cell carcinoma': 98, 'stasis edema': 99, 'stevens johnson syndrome': 100,
'striae': 101, 'sun damaged skin': 102, 'superficial spreading melanoma ssm':
103, 'syringoma': 104, 'telangiectases': 105, 'tick bite': 106, 'tuberous
sclerosis': 107, 'tungiasis': 108, 'urticaria': 109, 'urticaria pigmentosa':
110, 'vitiligo': 111, 'xanthomas': 112, 'xeroderma pigmentosum': 113}
--- group_map: {-1: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6}
--- number training:   64
--- number validation: 64
--- number testing:    64
-- Random state: 0
  0%|                                                          | 0/4 [00:00<?,
?it/s]/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
packages/torchvision/io/image.py:11: UserWarning: Failed to load image Python
extension: dlopen(/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
packages/torchvision/image.so, 0x0006): Library not loaded:
@rpath/libpng16.16.dylib
  Referenced from: <8E4D9E61-A0A3-30A7-B778-23E3E702B690>
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages/torchvision/image.so
  Reason: tried: '/opt/anaconda3/envs/dlh38env/lib/python3.8/lib-
dynload/../../libpng16.16.dylib' (no such file),
'/opt/anaconda3/envs/dlh38env/bin/../lib/libpng16.16.dylib' (no such file),
'/usr/local/lib/libpng16.16.dylib' (no such file), '/usr/lib/libpng16.16.dylib'
(no such file, not in dyld cache)
  warn(f"Failed to load image Python extension: {e}")
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
packages/torchvision/io/image.py:11: UserWarning: Failed to load image Python
extension: dlopen(/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
packages/torchvision/image.so, 0x0006): Library not loaded:
@rpath/libpng16.16.dylib
  Referenced from: <8E4D9E61-A0A3-30A7-B778-23E3E702B690>
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages/torchvision/image.so
  Reason: tried: '/opt/anaconda3/envs/dlh38env/lib/python3.8/lib-
dynload/../../libpng16.16.dylib' (no such file),
'/opt/anaconda3/envs/dlh38env/bin/../lib/libpng16.16.dylib' (no such file),
'/usr/local/lib/libpng16.16.dylib' (no such file), '/usr/lib/libpng16.16.dylib'
(no such file, not in dyld cache)
  warn(f"Failed to load image Python extension: {e}")
100%|                        | 4/4 [00:16<00:00,  4.07s/it]
  0%|                                                          | 0/4 [00:00<?,
?it/s]/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
packages/torchvision/io/image.py:11: UserWarning: Failed to load image Python
extension: dlopen(/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
```

```
packages/torchvision/image.so, 0x0006): Library not loaded:
@rpath/libpng16.16.dylib
  Referenced from: <8E4D9E61-A0A3-30A7-B778-23E3E702B690>
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages/torchvision/image.so
  Reason: tried: '/opt/anaconda3/envs/dlh38env/lib/python3.8/lib-
dynload/../../libpng16.16.dylib' (no such file),
'/opt/anaconda3/envs/dlh38env/bin/../lib/libpng16.16.dylib' (no such file),
'/usr/local/lib/libpng16.16.dylib' (no such file), '/usr/lib/libpng16.16.dylib'
(no such file, not in dyld cache)
  warn(f"Failed to load image Python extension: {e}")
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
packages/torchvision/io/image.py:11: UserWarning: Failed to load image Python
extension: dlopen(/opt/anaconda3/envs/dlh38env/lib/python3.8/site-
packages/torchvision/image.so, 0x0006): Library not loaded:
@rpath/libpng16.16.dylib
  Referenced from: <8E4D9E61-A0A3-30A7-B778-23E3E702B690>
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages/torchvision/image.so
  Reason: tried: '/opt/anaconda3/envs/dlh38env/lib/python3.8/lib-
dynload/../../libpng16.16.dylib' (no such file),
'/opt/anaconda3/envs/dlh38env/bin/../lib/libpng16.16.dylib' (no such file),
'/usr/local/lib/libpng16.16.dylib' (no such file), '/usr/lib/libpng16.16.dylib'
(no such file, not in dyld cache)
  warn(f"Failed to load image Python extension: {e}")
100%|                         | 4/4 [00:14<00:00,  3.73s/it]
Epoch=1 out of 1
train_loss=5.006352424621582
valid_loss=4.8282777070999146
    Saved checkpoint ==> 4.8282777070999146
    Finished training
    Best checkpoint epoch 0 - 4.8282777070999146
100%|                        | 64/64 [00:05<00:00, 11.01it/s]
100%|                        | 64/64 [00:05<00:00, 12.02it/s]
=-=-= total runtime: 44s =-=-=
```

5.5 Evaluation

5.5.1 Metrics Description

1. Marginal coverage – the number of times the true class is contained within prediction set, averaged over all samples.
2. Marginal cardinality – expected prediction set size, the average number of elements in the prediction set.
3. Subgroup coverage – marginal coverage per subgroup
4. Subgroup cardinality – marginal cardinality per subgroup

5.5.2 Evaluation code

In `.src/`, - conformal.py – contains functions for conformal predictions like the estimated score quantile q_hat and the confidence sets - metrics.py – contains functions used to report the metrics

like sensitivity, kappa, auroc, predictive parity etc. - platt.py – contains function for platt scaling to better calibrate the softmax scores - uncertainty.py – contains function to calculate uncertainty metrics like variance and entropy

mainly deal with these conformal metrics

[9]:
```
def parse_result(values):
    ret = {}
    image = values['meta']['image']
    label = values['meta']['label']
    subgroup = values['meta']['subgroup']
    ret['image'] = image
    ret['label'] = label
    ret['subgroup'] = subgroup

    #  group mc by class
    class_pred = collections.defaultdict(list)
    for k, v in values.items():
        if k.startswith('mc_'):
            for i, x in enumerate(v[0]):
                class_pred[i].append(x)

    for c, pred in class_pred.items():
        ret[f'pred_{c}'] = pred

    return ret
```

[10]:
```
# Load test paths dictionary with folder to filename
run_path = 'run_result/'
valid_paths = dict()
test_paths = dict()
for i in range(RUNS):
    valid_p = pathlib.Path(run_path + 'seed_' + str(i) + '/' ).
 ↪glob('valid-res*')
    #print(valid_p)
    for v in valid_p:
        valid_paths['seed_' + str(i)] = v
    test_p = pathlib.Path(run_path + 'seed_' + str(i) + '/' ).glob('test-res*')
    for t in test_p:
        test_paths['seed_' + str(i)] = t

label_map = dict(enumerate(sorted(skin_df.raw_label.unique())))
reversed_label_map = dict(zip(label_map.values(), label_map.keys()))
```

[11]:
```
# Loading stored training and validation dataset, from pickle file or the json␣
 ↪results
import pickle
```

```
try:
    with open(run_path+"valid_dfs.pickle","rb") as file:
        valid_dfs = pickle.load(file)
    print("Large dictionary loaded")
except:
    print("Pickle file not found. Processing from train result...")
    valid_dfs = {k: pd.DataFrame(list(map(parse_result, json.load(open(v)).
 ↪values()))) for k, v in valid_paths.items()}
    with open(run_path+"valid_dfs.pickle","wb") as file:
        pickle.dump(valid_dfs,file)
    print("Processing complete and pickle saved for valid_dfs.")

try:
    with open(run_path+"test_dfs.pickle","rb") as file:
        test_dfs = pickle.load(file)
    print("Large dictionary loaded")
except:
    print("Pickle file not found. Processing from train result...")
    test_dfs = {k: pd.DataFrame(list(map(parse_result, json.load(open(v)).
 ↪values()))) for k, v in test_paths.items()}
    with open(run_path+"test_dfs.pickle","wb") as file:
        pickle.dump(test_dfs,file)
    print("Processing complete and pickle saved for test_dfs.")
```

```
Large dictionary loaded
Large dictionary loaded
```

6 Results

6.1 Tables (and graphs) of results

Please refer to the graphs and the comments below.

6.1.1 Accuracy

[12]:
```
accuracy = []
for df in test_dfs.values():
    correct = 0
    for i, row in df[['label'] + [c for c in df.columns if c.
 ↪startswith('pred_')]].iterrows():
        label = row.label
        pred = np.argmax([c[0] for c in row[1:]])
        if label == pred: correct += 1
    accuracy.append(correct / len(df))
print(f'average accuracy {np.mean(accuracy):.2f} +/- {np.std(accuracy):.2f}')
```

```
average accuracy 0.25 +/- 0.00
```

[13]:
```
# number of classses in the Fitzpatrick 17k dataset
C = 114

def get_logits(row):
    return np.array([row[f'pred_{c}'][0] for c in range(C)]).tolist()
    # all_pred = [row[f'pred_{c}'] for c in range(C)]
    # return np.array(all_pred).mean(1).tolist()

for k, df in valid_dfs.items():
    df['logits'] = df.apply(lambda row: get_logits(row), axis=1)

for k, df in test_dfs.items():
    df['logits'] = df.apply(lambda row: get_logits(row), axis=1)
```

6.1.2 Apply platt scaling

To help better calibrate the output of the softmax function

[14]:
```
for k, df in valid_dfs.items():
    # learn temperature weight using validation set
    T = platt.get_platt_scaling(df.label.values, [x for x in df.logits.values])
    valid_logits = torch.tensor([x for x in df.logits.values])
    df['scores'] = torch.softmax(valid_logits / T, axis=1).tolist()

    # apply on test set
    test_df = test_dfs[k]
    test_logits = torch.tensor([x for x in test_df.logits.values])
    test_df['scores'] = torch.softmax(test_logits / T, axis=1).tolist()
```

6.1.3 Subgroups

The six fizpatrick skin types

[15]:
```
A = list(reversed(range(len(df.subgroup.unique()))))
A
```

[15]:
```
[6, 5, 4, 3, 2, 1, 0]
```

6.1.4 Miscoverage levels α

[16]:
```
ALPHAS = [round(x, 2) for x in np.arange(0.05, 0.55, 0.05)]
print(ALPHAS)
```

```
[0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
```

6.1.5 Conformal metrics

1. **Marginal coverage** – probability that the true class is contained within prediction set
2. **Marginal cardinality** - expected prediction set size

3. **Subgroup coverage** - marginal coverage per subgroup
4. **Subgroup cardinality** - marginal cardinality per subgroup

[17]:
```python
def get_coverage(labels: list, prediction_sets: list) -> float:
    k = len(labels)
    assert k
    correct = 0
    for label, pred in zip(labels, prediction_sets):
        correct += 1 if label in pred else 0
    return correct / k

def get_cardinality(prediction_sets: list) -> float:
    k = len(prediction_sets)
    assert k
    return sum([len(x) for x in prediction_sets]) / k

def get_subgroup_coverage(df, pred_col, label_col='label',␣
 ↪subgroup_col='subgroup', subgroups=A) -> dict:
    res = {}
    for a in subgroups:
        sub_df = df[df[subgroup_col] == a]
        res[a] = get_coverage(
            sub_df[label_col].values.tolist(),
            sub_df[pred_col].values.tolist(),
        )
    return res

def get_subgroup_cardinality(df, pred_col, subgroup_col='subgroup',␣
 ↪subgroups=A) -> dict:
    res = {}
    for a in subgroups:
        sub_df = df[df[subgroup_col] == a]
        res[a] = get_cardinality(sub_df[pred_col].values.tolist())
    return res
```

6.1.6 Helper functions

[18]:
```python
def aggregate_coverage(dfs, prefix, alphas=ALPHAS):
    # Collate prediction sets for all runs at different levels of coverage
    coverage_by_alpha = collections.defaultdict(list)
    for seed, df in dfs.items():
        for alpha in alphas:
            labels = [x for x in dfs[seed].label.values]
            prediction_sets = [x for x in dfs[seed][f'{prefix}_{alpha}'].values]
            coverage_by_alpha[alpha].append(get_coverage(labels,␣
 ↪prediction_sets))
```

```
    coverage_by_alpha = dict(coverage_by_alpha)
    alpha, coverage = zip(*coverage_by_alpha.items())
    coverage = np.array(coverage)
    coverage_mean = coverage.mean(1)
    coverage_std = coverage.std(1)
    return alpha, coverage_mean, coverage_std

def aggregate_cardinality(dfs, prefix, alphas=ALPHAS):
    cardinality_by_alpha = collections.defaultdict(list)

    for seed, df in dfs.items():
        for alpha in ALPHAS:
            labels = [x for x in dfs[seed].label.values]
            prediction_sets = [x for x in dfs[seed][f'{prefix}_{alpha}'].values]
            cardinality_by_alpha[alpha].append(get_cardinality(prediction_sets))

    cardinality_by_alpha = dict(cardinality_by_alpha)
    alpha, cardinality = zip(*cardinality_by_alpha.items())
    cardinality = np.array(cardinality)
    cardinality_mean = cardinality.mean(1)
    cardinality_std = cardinality.std(1)
    return alpha, cardinality_mean, cardinality_std
```

[19]:
```
# Helper functions for subgroup
def aggregate_coverage_subgroup(dfs, prefix, alphas=ALPHAS):
    # Collate prediction sets for all runs at different levels of coverage
    subgroup_coverage_by_alpha = collections.defaultdict(dict)
    for seed, df in dfs.items():
        for alpha in alphas:
            for sub, cov in get_subgroup_coverage(df, f'{prefix}_{alpha}').
 ↪items():
                if sub in subgroup_coverage_by_alpha[alpha]:
                    subgroup_coverage_by_alpha[alpha][sub] += [cov]
                else:
                    subgroup_coverage_by_alpha[alpha][sub] = [cov]

    alpha, subgroup_cov = zip(*subgroup_coverage_by_alpha.items())

    coverage_mean = collections.defaultdict(list)
    coverage_std = collections.defaultdict(list)

    for res in subgroup_cov:
        for group, cov in res.items():
            cov = np.array(cov)
            coverage_mean[group].append(cov.mean())
            coverage_std[group].append(cov.std())
```

```
    coverage_mean = dict(coverage_mean)
    coverage_std = dict(coverage_std)

    return alpha, coverage_mean, coverage_std


def aggregate_cardinality_subgroup(dfs, prefix, alphas=ALPHAS):
    subgroup_cardinality_by_alpha = collections.defaultdict(dict)

    for seed, df in dfs.items():
        for alpha in ALPHAS:
            for sub, card in get_subgroup_cardinality(df, f'{prefix}_{alpha}').
 ↪items():
                if sub in subgroup_cardinality_by_alpha[alpha]:
                    subgroup_cardinality_by_alpha[alpha][sub] += [card]
                else:
                    subgroup_cardinality_by_alpha[alpha][sub] = [card]

    subgroup_cardinality_by_alpha = dict(subgroup_cardinality_by_alpha)
    alpha, cardinality = zip(*subgroup_cardinality_by_alpha.items())

    cardinality_mean = collections.defaultdict(list)
    cardinality_std = collections.defaultdict(list)

    for res in cardinality:
        for group, card in res.items():
            card = np.array(card)
            cardinality_mean[group].append(card.mean())
            cardinality_std[group].append(card.std())

    cardinality_mean = dict(cardinality_mean)
    cardinality_std = dict(cardinality_std)

    return alpha, cardinality_mean, cardinality_std
```

6.1.7 Naive prediction sets

- Platt scaling on logits
- Form set by adding elements from sorted softmax scores until cumulative sum exceeds $1 - \alpha$

[20]:
```
naive_qhat = {}
for k, df in valid_dfs.items():
    scores = [x for x in df.scores.values]
    labels = [x for x in df.label.values]
    s = torch.tensor([1 - s[l] for s, l in zip(scores, labels)])
    n = len(df)
    for alpha in ALPHAS:
```

```
        p = np.ceil((n + 1) * (1 - alpha)) / n
        naive_qhat[alpha] = torch.quantile(s, p).item()

for k, df in test_dfs.items():
    scores = np.array([x for x in df.scores.values])
    for alpha in ALPHAS:
        df[f'naive_{alpha}'] = [np.nonzero(s > (1 - naive_qhat[alpha]))[0] for
 ↪s in scores]
```

[21]:
```
columns = [f'naive_{alpha}' for alpha in ALPHAS]
test_dfs['seed_0'][columns].head()
```

[21]:

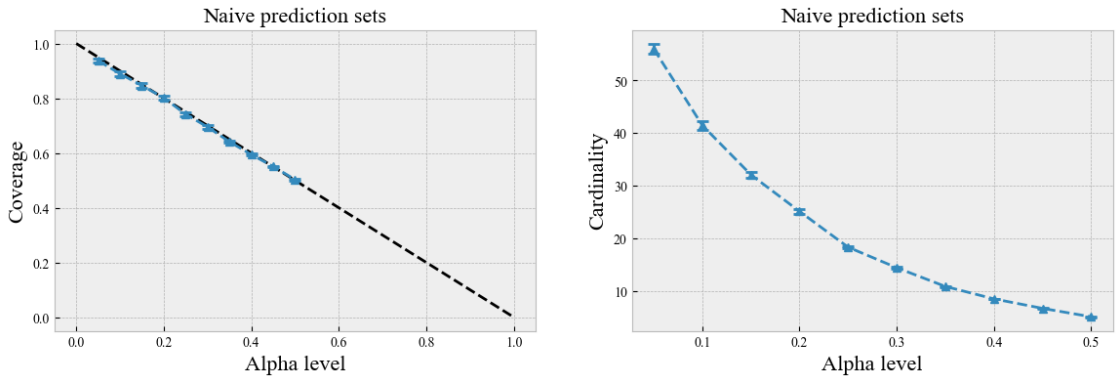
	naive_0.05	naive_0.1	naive_0.15	naive_0.2	naive_0.25
0	[0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 14, 16…	[0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 19, 20…	[1, 6, 8, 10, 12, 14, 19, 20, 22, 23, 24, 31, …	[1, 6, 10, 12, 14, 19, 20, 22, 24, 31, 32, 37,…	[6, 10, 14, 20, 22, 24, 31, 32, 37, 41, 51, 53…
1	[0, 1, 3, 4, 5, 6, 7, 8, 12, 13, 14, 15, 16, 1…	[1, 3, 4, 5, 6, 7, 8, 12, 13, 14, 15, 16, 17, …	[4, 5, 6, 7, 8, 12, 14, 15, 16, 17, 22, 23, 25…	[4, 5, 6, 7, 12, 14, 15, 16, 17, 22, 23, 25, 2…	[4, 5, 6, 7, 12, 15, 22, 23, 25, 28, 44, 48, 5…
2	[5, 6, 7, 8, 11, 16, 17, 18, 19, 23, 25, 26, 2…	[5, 6, 8, 11, 17, 18, 25, 27, 28, 30, 34, 38, …	[5, 8, 11, 18, 25, 27, 28, 34, 38, 48, 52, 59,…	[5, 11, 18, 27, 28, 48, 52, 59, 60, 61, 67, 68…	[5, 11, 27, 28, 48, 52, 59, 80, 82, 83, 85, 91…
3	[0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 14, 15, 17, 19…	[0, 1, 2, 5, 6, 7, 8, 10, 14, 15, 17, 19, 21, …	[0, 1, 2, 5, 6, 7, 8, 14, 15, 19, 25, 31, 34, …	[0, 1, 5, 8, 14, 15, 19, 34, 39, 42, 45, 47, 4…	[0, 1, 5, 8, 39, 42, 45, 47, 49, 51, 56, 58, 6…
4	[1, 3, 4, 5, 6, 8, 9, 11, 13, 15, 21, 22, 25, …	[1, 3, 4, 5, 6, 8, 11, 13, 15, 21, 22, 31, 32,…	[3, 5, 6, 8, 11, 13, 15, 21, 22, 32, 34, 36, 4…	[3, 5, 6, 8, 11, 15, 21, 22, 34, 36, 48, 51, 5…	[3, 5, 6, 8, 15, 21, 22, 34, 51, 54, 60, 63, 6…

```
                                          naive_0.3  \
0  [6, 10, 20, 24, 31, 32, 37, 41, 51, 53, 57, 58…
1  [4, 5, 6, 12, 15, 22, 23, 25, 28, 48, 50, 51, …
2    [5, 11, 27, 28, 48, 52, 82, 83, 85, 91, 92, 98]
3  [1, 8, 39, 42, 45, 49, 51, 56, 58, 68, 80, 81,…
4  [5, 6, 8, 21, 34, 51, 54, 63, 64, 65, 68, 86, …

                                         naive_0.35  \
0  [6, 10, 20, 24, 31, 37, 41, 53, 57, 66, 69, 79…
1  [6, 12, 22, 23, 25, 28, 50, 51, 60, 72, 86, 92…
2              [27, 28, 48, 52, 82, 83, 85, 91, 92]
3  [1, 8, 39, 45, 49, 56, 58, 68, 80, 81, 86, 98,…
4  [5, 6, 8, 21, 34, 54, 63, 64, 65, 68, 86, 87, 98]

                                          naive_0.4  \
0  [6, 10, 20, 24, 31, 53, 57, 91, 93, 98, 105, 110]
1  [6, 12, 22, 23, 25, 28, 50, 51, 60, 72, 86, 92…
2              [27, 28, 48, 52, 82, 83, 85, 91, 92]
3       [1, 8, 45, 49, 56, 58, 68, 80, 81, 86, 113]
4                  [5, 6, 8, 54, 63, 64, 68, 86, 98]

                                  naive_0.45                    naive_0.5
0                    [6, 10, 53, 57, 93, 105]             [10, 53, 57, 105]
1  [6, 12, 22, 25, 28, 50, 51, 60, 72, 86, 93]           [6, 28, 51, 86, 93]
2            [27, 28, 48, 52, 83, 85, 91, 92]  [27, 28, 48, 83, 85, 91, 92]
3          [1, 8, 45, 56, 58, 68, 80, 81, 86]          [58, 68, 80, 81, 86]
4                    [5, 6, 8, 54, 63, 68, 98]     [5, 6, 8, 54, 63, 68, 98]
```

[22]:
```
alpha, naive_coverage_mean, naive_coverage_std = aggregate_coverage(test_dfs,␣
↪prefix='naive')
alpha, naive_cardinality_mean, naive_cardinality_std =␣
↪aggregate_cardinality(test_dfs, prefix='naive')
```

[23]:
```
fontsize=16
fig, ax = plt.subplots(ncols=2, figsize=(14, 4))
ax[0].plot([0, 1], [1, 0], '--', c='k', label='Marginal Coverage')
ax[0].errorbar(
    alpha, naive_coverage_mean, yerr=naive_coverage_std,
    marker='^', ls='--', capthick=2, capsize=4,
)
ax[0].set_title('Naive prediction sets', fontsize=fontsize)
ax[0].set_xlabel('Alpha level', fontsize=fontsize)
ax[0].set_ylabel('Coverage', fontsize=fontsize)

ax[1].errorbar(
    alpha, naive_cardinality_mean, yerr=naive_cardinality_std,
    marker='^', ls='--', capthick=2, capsize=4,
```

```
)
ax[1].set_title('Naive prediction sets', fontsize=fontsize)
ax[1].set_xlabel('Alpha level', fontsize=fontsize)
ax[1].set_ylabel('Cardinality', fontsize=fontsize)
plt.show()
```



Coverage rate v.s. alpha level and prediction set size v.s. alpha level; both for naive implementation

6.1.8 Adaptive prediction set (APS)

- Calibration set for correct marginal coverage

[24]:
```
importlib.reload(conformal)
```

[24]:
```
<module 'src.conformal' from
'/Users/tianhaoluo/DLH/CS598DLHFinal/src/conformal.py'>
```

[25]:
```
# calibrate
aps_qhat = collections.defaultdict(dict)
for k, df in valid_dfs.items():
    scores = [x for x in df.scores.values]
    labels = [x for x in df.label.values]
    for alpha in ALPHAS:
        qhat = conformal.get_q_hat(scores, labels, alpha=alpha)
        aps_qhat[k][alpha] = qhat.item()

aps_qhat = dict(aps_qhat)
```

[26]:
```
print(aps_qhat['seed_0'][0.1])
```

```
0.9519096612930298
```

[27]:
```
# inference
for k, df in test_dfs.items():
    scores = [x for x in df.scores.values]
```

```
    for alpha in ALPHAS:
        qhat = aps_qhat[k][alpha]
        df[f'aps_{alpha}'] = conformal.conformal_inference(scores, qhat)
```

[28]:
```
columns = [f'aps_{alpha}' for alpha in ALPHAS]
test_dfs['seed_0'][columns].head()
```

[28]:

```
                                              aps_0.05  \
0  [53, 10, 105, 57, 93, 6, 20, 31, 110, 91, 24, …
1  [6, 93, 28, 86, 51, 72, 22, 25, 50, 60, 12, 23…
2  [27, 48, 91, 28, 85, 92, 83, 52, 82, 11, 98, 5…
3  [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
4  [6, 8, 98, 63, 68, 5, 54, 86, 64, 87, 21, 34, …

                                               aps_0.1  \
0  [53, 10, 105, 57, 93, 6, 20, 31, 110, 91, 24, …
1  [6, 93, 28, 86, 51, 72, 22, 25, 50, 60, 12, 23…
2  [27, 48, 91, 28, 85, 92, 83, 52, 82, 11, 98, 5…
3  [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
4  [6, 8, 98, 63, 68, 5, 54, 86, 64, 87, 21, 34, …

                                              aps_0.15  \
0  [53, 10, 105, 57, 93, 6, 20, 31, 110, 91, 24, …
1  [6, 93, 28, 86, 51, 72, 22, 25, 50, 60, 12, 23…
2  [27, 48, 91, 28, 85, 92, 83, 52, 82, 11, 98, 5…
3  [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
4  [6, 8, 98, 63, 68, 5, 54, 86, 64, 87, 21, 34, …

                                               aps_0.2  \
0  [53, 10, 105, 57, 93, 6, 20, 31, 110, 91, 24, …
1  [6, 93, 28, 86, 51, 72, 22, 25, 50, 60, 12, 23…
2  [27, 48, 91, 28, 85, 92, 83, 52, 82, 11, 98, 5…
3  [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
4  [6, 8, 98, 63, 68, 5, 54, 86, 64, 87, 21, 34, …

                                              aps_0.25  \
0  [53, 10, 105, 57, 93, 6, 20, 31, 110, 91, 24, …
1  [6, 93, 28, 86, 51, 72, 22, 25, 50, 60, 12, 23…
2  [27, 48, 91, 28, 85, 92, 83, 52, 82, 11, 98, 5…
3  [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
4  [6, 8, 98, 63, 68, 5, 54, 86, 64, 87, 21, 34, …

                                               aps_0.3  \
0  [53, 10, 105, 57, 93, 6, 20, 31, 110, 91, 24, …
1  [6, 93, 28, 86, 51, 72, 22, 25, 50, 60, 12, 23…
2  [27, 48, 91, 28, 85, 92, 83, 52, 82, 11, 98, 5…
3  [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
```

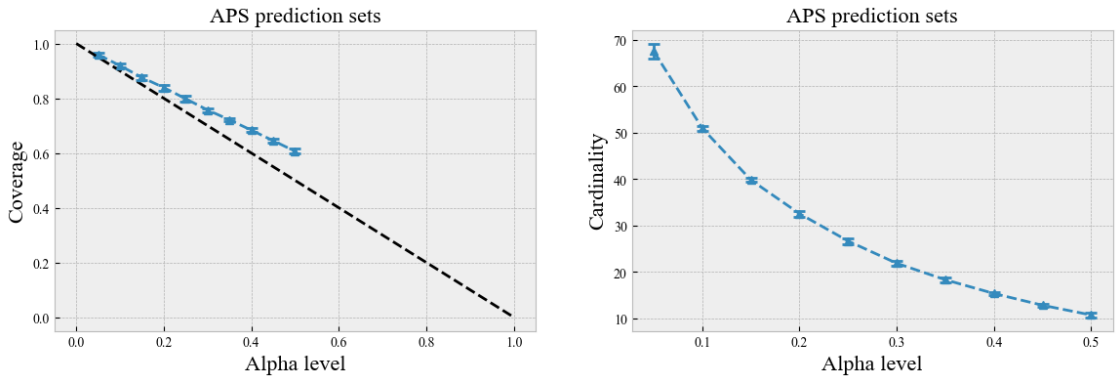
```
4  [6, 8, 98, 63, 68, 5, 54, 86, 64, 87, 21, 34, …
```

```
                                                aps_0.35  \
0  [53, 10, 105, 57, 93, 6, 20, 31, 110, 91, 24, …
1  [6, 93, 28, 86, 51, 72, 22, 25, 50, 60, 12, 23…
2    [27, 48, 91, 28, 85, 92, 83, 52, 82, 11, 98, 5]
3  [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
4  [6, 8, 98, 63, 68, 5, 54, 86, 64, 87, 21, 34, …

                                                 aps_0.4  \
0  [53, 10, 105, 57, 93, 6, 20, 31, 110, 91, 24, …
1  [6, 93, 28, 86, 51, 72, 22, 25, 50, 60, 12, 23…
2              [27, 48, 91, 28, 85, 92, 83, 52, 82]
3  [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
4  [6, 8, 98, 63, 68, 5, 54, 86, 64, 87, 21, 34, 65]

                                                aps_0.45  \
0  [53, 10, 105, 57, 93, 6, 20, 31, 110, 91, 24, …
1  [6, 93, 28, 86, 51, 72, 22, 25, 50, 60, 12, 23…
2                  [27, 48, 91, 28, 85, 92, 83, 52]
3  [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
4              [6, 8, 98, 63, 68, 5, 54, 86, 64, 87]

                                                  aps_0.5
0  [53, 10, 105, 57, 93, 6, 20, 31, 110, 91, 24, …
1  [6, 93, 28, 86, 51, 72, 22, 25, 50, 60, 12, 23…
2                      [27, 48, 91, 28, 85, 92, 83]
3  [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
4                  [6, 8, 98, 63, 68, 5, 54, 86, 64]
```

[29]:
```
alpha, aps_coverage_mean, aps_coverage_std = aggregate_coverage(test_dfs,␣
 ↪prefix='aps')
alpha, aps_cardinality_mean, aps_cardinality_std =␣
 ↪aggregate_cardinality(test_dfs, prefix='aps')
```

[30]:
```
fontsize=16
fig, ax = plt.subplots(ncols=2, figsize=(14, 4))
ax[0].errorbar(
    alpha, aps_coverage_mean, yerr=aps_coverage_std,
    marker='^', ls='--', capthick=2, capsize=4,
)
ax[0].plot([0, 1], [1, 0], '--', c='k', label='Marginal Coverage')
ax[0].set_title('APS prediction sets', fontsize=fontsize)
ax[0].set_xlabel('Alpha level', fontsize=fontsize)
ax[0].set_ylabel('Coverage', fontsize=fontsize)

ax[1].errorbar(
```

```
    alpha, aps_cardinality_mean, yerr=aps_cardinality_std,
    marker='^', ls='--', capthick=2, capsize=4,
)
ax[1].set_title('APS prediction sets', fontsize=fontsize)
ax[1].set_xlabel('Alpha level', fontsize=fontsize)
ax[1].set_ylabel('Cardinality', fontsize=fontsize)
plt.show()
```



6.1.9 Group APS (GAPS)

Conformal calibration for each subgroup

The dataset will be a tuple containing the training example, group attribute and targeting class. The scoring function sorts the classes according to their softmax score in descing order for each exmaple and outputs the cumulative sum until the softmax score of the true class in reached.

[31]:
```
# calibrate
gaps_qhat = collections.defaultdict(dict)

for k, df in valid_dfs.items():
    alpha_qhat = collections.defaultdict(dict)
    for alpha in ALPHAS:
        group_qhat = collections.defaultdict(dict)
        for a in A:
            group_df = df[df.subgroup == a]

            scores = [x for x in group_df.scores.values]
            labels = [x for x in group_df.label.values]
            qhat = conformal.get_q_hat(scores, labels, alpha=alpha)
            gaps_qhat[k][alpha] = qhat.item()

            group_qhat[a] = qhat

        alpha_qhat[alpha] = dict(group_qhat)
```

```
    gaps_qhat[k] = dict(alpha_qhat)

gaps_qhat = dict(gaps_qhat)
```

[32]:
```
# inference

for k, df in test_dfs.items():
    temp = []
    scores = np.array([x for x in df.scores.values])
    index, ordered, cumsum = conformal.sort_sum(scores)
    for a in A:
        group_df = df[df.subgroup == a].copy()

        # APS
        scores = [x for x in group_df.scores.values]
        for alpha in ALPHAS:
            qhat = gaps_qhat[k][alpha][a]
            group_df[f'gaps_{alpha}'] = conformal.conformal_inference(scores,␣
 ↪qhat)

        temp.append(group_df)
    test_dfs[k] = pd.concat(temp)
```

[33]:
```
# columns = [f'gaps_{alpha}' for alpha in ALPHAS]
columns = [f'gaps_{alpha}' for alpha in ALPHAS]
test_dfs['seed_0'][columns].head()
```

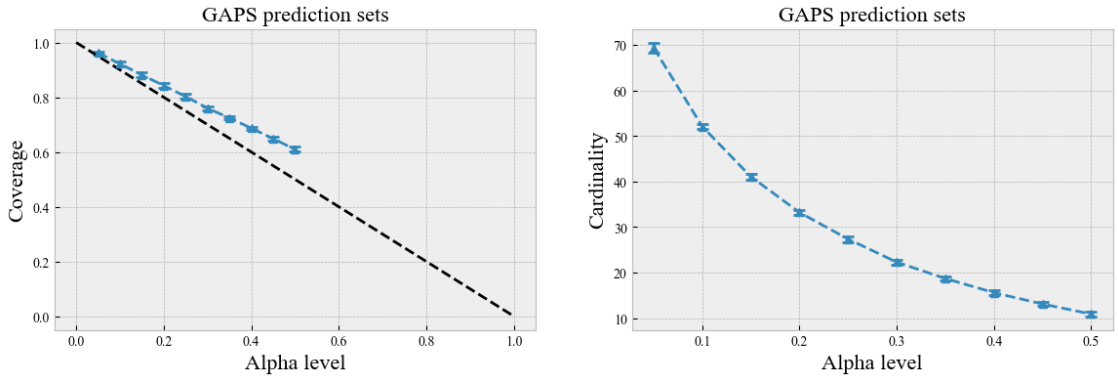
[33]:
```
                                              gaps_0.05  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                               gaps_0.1  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                              gaps_0.15  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
```

```
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                 gaps_0.2  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                gaps_0.25  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                 gaps_0.3  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                gaps_0.35  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33           [51, 68, 86, 32, 80, 113, 98, 42, 91, 49]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                 gaps_0.4  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33                                  [51, 68, 86, 32, 80]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                gaps_0.45  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33                                       [51, 68, 86, 32]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                  gaps_0.5
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33                                          [51, 68, 86]
```

```
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …
```

[34]:
```
alpha, gaps_coverage_mean, gaps_coverage_std = aggregate_coverage(test_dfs,␣
 ↪prefix='gaps')
alpha, gaps_cardinality_mean, gaps_cardinality_std =␣
 ↪aggregate_cardinality(test_dfs, prefix='gaps')
```

[35]:
```
fontsize=16
fig, ax = plt.subplots(ncols=2, figsize=(14, 4))
ax[0].errorbar(
    alpha, gaps_coverage_mean, yerr=gaps_coverage_std,
    marker='^', ls='--', capthick=2, capsize=4,
)
ax[0].plot([0, 1], [1, 0], '--', c='k', label='Marginal Coverage')
ax[0].set_title('GAPS prediction sets', fontsize=fontsize)
ax[0].set_xlabel('Alpha level', fontsize=fontsize)
ax[0].set_ylabel('Coverage', fontsize=fontsize)

ax[1].errorbar(
    alpha, gaps_cardinality_mean, yerr=gaps_cardinality_std,
    marker='^', ls='--', capthick=2, capsize=4,
)
ax[1].set_title('GAPS prediction sets', fontsize=fontsize)
ax[1].set_xlabel('Alpha level', fontsize=fontsize)
ax[1].set_ylabel('Cardinality', fontsize=fontsize)
plt.show()
```



6.1.10 Regularized adaptive prediction set (RAPS)

- Calibration set for correct marginal coverage
- Parameterized regulaization penality λ and K

[36]:
```
K_REG = 0
LAMBDA = 1e-4

penalty = np.zeros((1, C))
penalty[:, K_REG:] += LAMBDA

# randomized
RAND = True

# allow zero sets
ZERO = True
```

[37]:
```
# calibrate
raps_qhat = collections.defaultdict(dict)
for k, df in valid_dfs.items():
    scores = np.array([x for x in df.scores.values])
    labels = np.array([x for x in df.label.values])
    index, ordered, cumsum = conformal.sort_sum(scores)
    for alpha in ALPHAS:
        qhat = conformal.raps_calibrate(
            scores, labels, index, ordered, cumsum,
            penalty, randomized=RAND, allow_zero_sets=ZERO, alpha=alpha
        )
        raps_qhat[k][alpha] = qhat

raps_qhat = dict(raps_qhat)
```

[38]:
```
print(raps_qhat['seed_0'][0.10])
```

```
0.953587291286114
```

[39]:
```
# inference
for k, df in test_dfs.items():
    scores = np.array([x for x in df.scores.values])
    index, ordered, cumsum = conformal.sort_sum(scores)
    for alpha in ALPHAS:
        qhat = raps_qhat[k][alpha]
        df[f'raps_{alpha}'] = conformal.raps_predict(
            scores, qhat, index, ordered, cumsum,
            penalty, randomized=RAND, allow_zero_sets=ZERO,
        )
```

[40]:
```
columns = [f'raps_{alpha}' for alpha in ALPHAS]
test_dfs['seed_0'][columns].head()
```

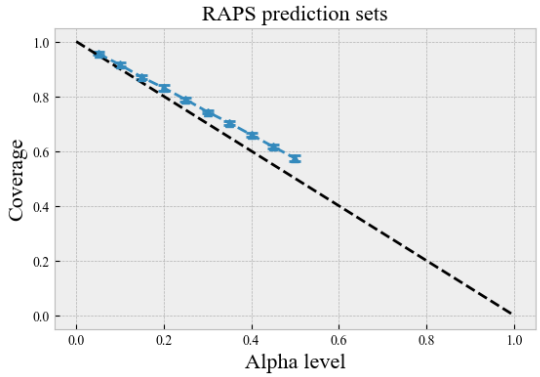
[40]:
```
                                            raps_0.05  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
```

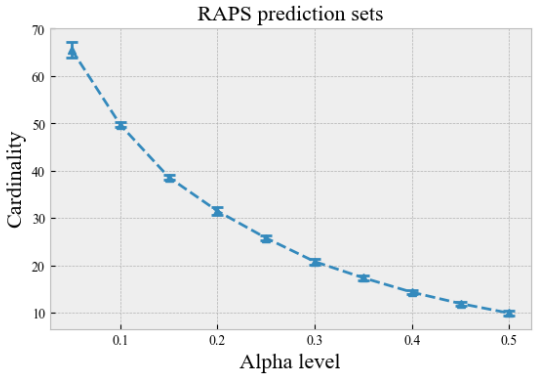
```
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                raps_0.1  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                               raps_0.15  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                raps_0.2  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                               raps_0.25  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                raps_0.3  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33               [51, 68, 86, 32, 80, 113, 98, 42, 91]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                               raps_0.35  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33                                 [51, 68, 86, 32, 80]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                raps_0.4  \
```

```
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33                                          [51, 68, 86]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                              raps_0.45  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33                                          [51, 68, 86]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                               raps_0.5
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33                                           [51, 68]
61       [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100]
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …
```

[41]:
```
alpha, raps_coverage_mean, raps_coverage_std = aggregate_coverage(test_dfs,␣
 ↪prefix='raps')
alpha, raps_cardinality_mean, raps_cardinality_std =␣
 ↪aggregate_cardinality(test_dfs, prefix='raps')
```

[42]:
```
fontsize=16
fig, ax = plt.subplots(ncols=2, figsize=(14, 4))
ax[0].errorbar(
    alpha, raps_coverage_mean, yerr=raps_coverage_std,
    marker='^', ls='--', capthick=2, capsize=4,
)
ax[0].plot([0, 1], [1, 0], '--', c='k', label='Marginal Coverage')
ax[0].set_title('RAPS prediction sets', fontsize=fontsize)
ax[0].set_xlabel('Alpha level', fontsize=fontsize)
ax[0].set_ylabel('Coverage', fontsize=fontsize)

ax[1].errorbar(
    alpha, raps_cardinality_mean, yerr=raps_cardinality_std,
    marker='^', ls='--', capthick=2, capsize=4,
)
ax[1].set_title('RAPS prediction sets', fontsize=fontsize)
ax[1].set_xlabel('Alpha level', fontsize=fontsize)
ax[1].set_ylabel('Cardinality', fontsize=fontsize)
plt.show()
```





6.1.11 Group RAPS (GRAPS)

[43]:
```
# calibrate
graps_qhat = collections.defaultdict(dict)

for k, df in valid_dfs.items():
    alpha_qhat = collections.defaultdict(dict)
    for alpha in ALPHAS:
        group_qhat = collections.defaultdict(dict)
        for a in A:
            group_df = df[df.subgroup == a]

            scores = np.array([x for x in group_df.scores.values])
            labels = np.array([x for x in group_df.label.values])
            index, ordered, cumsum = conformal.sort_sum(scores)
            qhat = conformal.raps_calibrate(
                scores, labels, index, ordered, cumsum,
                penalty, randomized=RAND, allow_zero_sets=ZERO, alpha=alpha
            )

            group_qhat[a] = qhat

        alpha_qhat[alpha] = dict(group_qhat)

    graps_qhat[k] = dict(alpha_qhat)

graps_qhat = dict(graps_qhat)
```

[44]:
```
# inference

for k, df in test_dfs.items():
    temp = []
    scores = np.array([x for x in df.scores.values])
```

```
    index, ordered, cumsum = conformal.sort_sum(scores)
    for a in A:
        group_df = df[df.subgroup == a].copy()

        # RAPS
        scores = np.array([x for x in group_df.scores.values])
        for alpha in ALPHAS:
            qhat = graps_qhat[k][alpha][a]
            index, ordered, cumsum = conformal.sort_sum(scores)
            group_df[f'graps_{alpha}'] = conformal.raps_predict(
                scores, qhat.item(), index, ordered, cumsum,
                penalty, randomized=RAND, allow_zero_sets=ZERO,
            )

        temp.append(group_df)
    test_dfs[k] = pd.concat(temp)
```

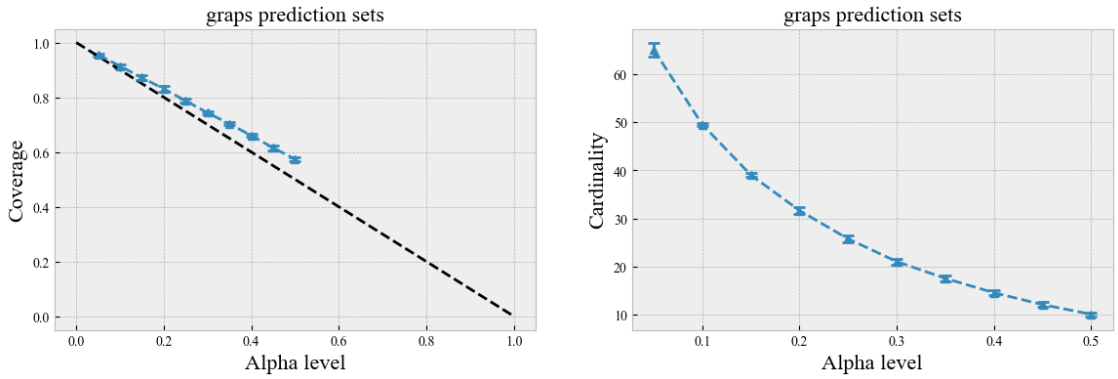
[45]:
```
columns = [f'graps_{alpha}' for alpha in ALPHAS]
test_dfs['seed_0'][columns].head()
```

[45]:
```
                                              graps_0.05  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                               graps_0.1  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                              graps_0.15  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                               graps_0.2  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …
```

```
                                                graps_0.25  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33   [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, …
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                 graps_0.3  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33    [51, 68, 86, 32, 80, 113, 98, 42, 91, 49, 48, 8]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                graps_0.35  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33               [51, 68, 86, 32, 80, 113, 98, 42, 91]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                 graps_0.4  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33                                    [51, 68, 86, 32]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                graps_0.45  \
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33                                       [51, 68, 86]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, …
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …

                                                  graps_0.5
3    [80, 86, 68, 81, 58, 45, 1, 8, 56, 113, 49, 98…
33                                       [51, 68, 86]
61   [91, 2, 51, 43, 74, 59, 4, 13, 109, 105, 100, 95]
69   [68, 98, 61, 76, 66, 73, 38, 6, 8, 48, 30, 92,…
118  [92, 6, 74, 19, 51, 86, 60, 100, 105, 22, 68, …
```

[46]:
```
alpha, graps_coverage_mean, graps_coverage_std = aggregate_coverage(test_dfs,␣
 ↪prefix='graps')
alpha, graps_cardinality_mean, graps_cardinality_std =␣
 ↪aggregate_cardinality(test_dfs, prefix='graps')
```

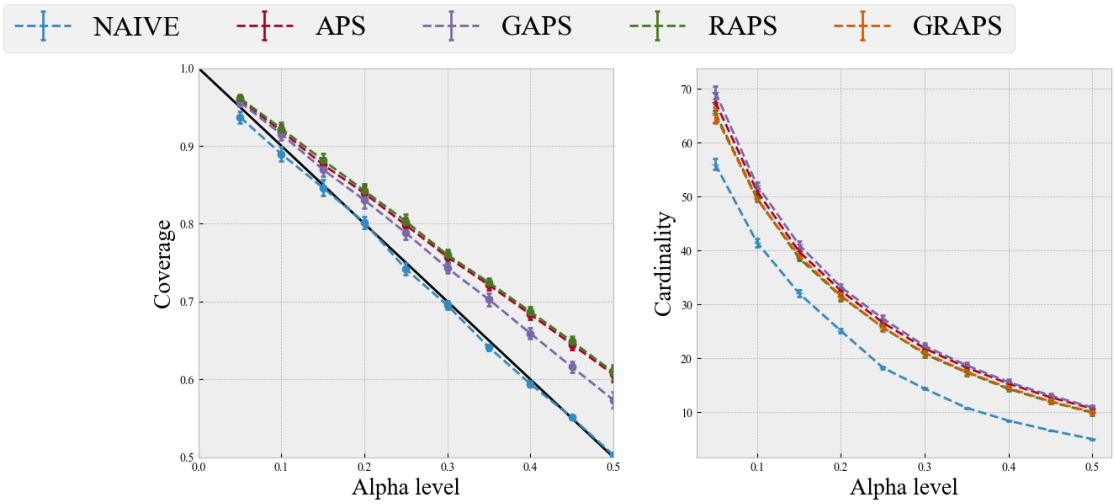
[47]:
```
fontsize=16
fig, ax = plt.subplots(ncols=2, figsize=(14, 4))
ax[0].errorbar(
    alpha, graps_coverage_mean, yerr=graps_coverage_std,
    marker='^', ls='--', capthick=2, capsize=4,
)
ax[0].plot([0, 1], [1, 0], '--', c='k', label='Marginal Coverage')
ax[0].set_title('graps prediction sets', fontsize=fontsize)
ax[0].set_xlabel('Alpha level', fontsize=fontsize)
ax[0].set_ylabel('Coverage', fontsize=fontsize)

ax[1].errorbar(
    alpha, graps_cardinality_mean, yerr=graps_cardinality_std,
    marker='^', ls='--', capthick=2, capsize=4,
)
ax[1].set_title('graps prediction sets', fontsize=fontsize)
ax[1].set_xlabel('Alpha level', fontsize=fontsize)
ax[1].set_ylabel('Cardinality', fontsize=fontsize)
plt.show()
```



6.1.12 Overall comparison

[48]:
```
capsize = 2
fontsize=20
capthick=2
lw = 2
marker='o'
fig, ax = plt.subplots(ncols=2, figsize=(14, 6))
ax[0].set_xlim(0, 0.5)
ax[0].set_ylim(0.5, 1)
ax[0].plot([1, 0], [0, 1], c='k', label='Ideal Coverage')
ax[0].errorbar(
    alpha, naive_coverage_mean, yerr=naive_coverage_std, lw=lw,
```

```
    marker=marker, ls='--', capthick=capthick, capsize=capsize, label='NAIVE',
)
ax[0].errorbar(
    alpha, aps_coverage_mean, yerr=aps_coverage_std, lw=lw,
    marker=marker, ls='--', capthick=capthick, capsize=capsize, label='APS',
)
ax[0].errorbar(
    alpha, raps_coverage_mean, yerr=raps_coverage_std, lw=lw,
    marker=marker, ls='--', capthick=capthick, capsize=capsize, label='RAPS',
)
ax[0].errorbar(
    alpha, gaps_coverage_mean , yerr=gaps_coverage_std, lw=lw,
    marker=marker, ls='--', capthick=capthick, capsize=capsize, label='GAPS',
)
ax[0].set_xlabel('Alpha level', fontsize=fontsize)
ax[0].set_ylabel('Coverage', fontsize=fontsize)

ax[1].errorbar(
    alpha, naive_cardinality_mean, yerr=naive_cardinality_std, lw=lw,
    marker='_', ls='--', capthick=capthick, capsize=capsize, label='NAIVE',
)
ax[1].errorbar(
    alpha, aps_cardinality_mean, yerr=aps_cardinality_std, lw=lw,
    marker='_', ls='--', capthick=capthick, capsize=capsize, label='APS',
)
ax[1].errorbar(
    alpha, gaps_cardinality_mean , yerr=gaps_cardinality_std, lw=lw,
    marker='_', ls='--', capthick=capthick, capsize=capsize, label='GAPS',
)
ax[1].errorbar(
    alpha, raps_cardinality_mean, yerr=raps_cardinality_std, lw=lw,
    marker='_', ls='--', capthick=capthick, capsize=capsize, label='RAPS',
)
ax[1].errorbar(
    alpha, graps_cardinality_mean , yerr=graps_cardinality_std, lw=lw,
    marker='_', ls='--', capthick=capthick, capsize=capsize, label='GRAPS',
)
ax[1].set_xlabel('Alpha level', fontsize=fontsize)
ax[1].set_ylabel('Cardinality', fontsize=fontsize)
plt.legend(fontsize=fontsize + 4, bbox_to_anchor=(0.8, 1.2), ncol=5)
plt.savefig(fig_dir / 'fitz-compare.png')
plt.show()
```



6.1.13 Subgroup comparison

[49]:
```
alpha, naive_subgroup_coverage_mean, naive_subgroup_coverage_std =␣
 ↪aggregate_coverage_subgroup(
    test_dfs, prefix='naive'
)
alpha, naive_subgroup_cardinality_mean, naive_subgroup_cardinality_std =␣
 ↪aggregate_cardinality_subgroup(
    test_dfs, prefix='naive'
)
alpha, aps_subgroup_coverage_mean, aps_subgroup_coverage_std =␣
 ↪aggregate_coverage_subgroup(
    test_dfs, prefix='aps'
)
alpha, aps_subgroup_cardinality_mean, aps_subgroup_cardinality_std =␣
 ↪aggregate_cardinality_subgroup(
    test_dfs, prefix='aps'
)
alpha, raps_subgroup_coverage_mean, raps_subgroup_coverage_std =␣
 ↪aggregate_coverage_subgroup(
    test_dfs, prefix='raps'
)
alpha, raps_subgroup_cardinality_mean, raps_subgroup_cardinality_std =␣
 ↪aggregate_cardinality_subgroup(
    test_dfs, prefix='raps'
)
alpha, gaps_subgroup_coverage_mean, gaps_subgroup_coverage_std =␣
 ↪aggregate_coverage_subgroup(
    test_dfs, prefix='gaps'
```

```
)
alpha, gaps_subgroup_cardinality_mean, gaps_subgroup_cardinality_std =␣
 ↪aggregate_cardinality_subgroup(
    test_dfs, prefix='gaps'
)
alpha, graps_subgroup_coverage_mean, graps_subgroup_coverage_std =␣
 ↪aggregate_coverage_subgroup(
    test_dfs, prefix='graps'
)
alpha, graps_subgroup_cardinality_mean, graps_subgroup_cardinality_std =␣
 ↪aggregate_cardinality_subgroup(
    test_dfs, prefix='graps'
)
```

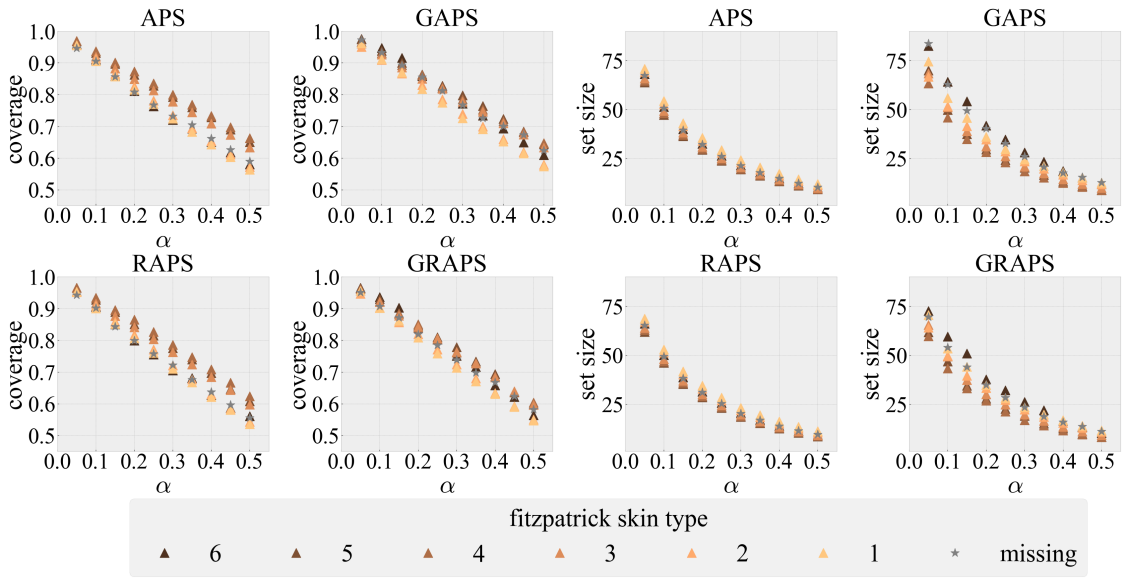
[50]:
```
fontsize=54
labelsize=48
fig, ax = plt.subplots(nrows=2, ncols=4, figsize=(34, 18), sharex=False,␣
 ↪sharey=False)
ls=''
lw=6
marker = '^'
ms = 20
alpha=1.0
xticks = np.arange(0.0, 0.51, 0.1)
yticks = np.arange(0.5, 1.01, 0.1)
xlim = (0, 0.55)
ylim = (0.45, 1.0)


# ax[0, 0].plot([0.1, 1], [1, 0], '--', c='k')
for i, (k, v) in enumerate(aps_subgroup_coverage_mean.items()):
    ax[0, 0].plot(ALPHAS, v, label=f'{"missing" if k == 0 else k}', ls=ls,␣
 ↪lw=lw, alpha=alpha, marker='*' if k == 0 else marker, ms=ms, c='gray' if k␣
 ↪== 0 else SKIN_COLORS[k-1])
# ax[0, 0].axvline(x=0.05, color="black", linestyle='--', lw=lw)
# ax[0, 0].text(0.08, 0.11, '95% confidence level', rotation=0,␣
 ↪fontsize=fontsize)
ax[0, 0].set_title('APS', fontsize=fontsize )
ax[0, 0].set_xlim(*xlim)
ax[0, 0].set_ylim(*ylim)
ax[0, 0].set_xlabel(r'$\alpha$', fontsize=fontsize)
ax[0, 0].set_ylabel('coverage', fontsize=fontsize)
ax[0, 0].set_xticks(xticks)
ax[0, 0].set_yticks(yticks)
ax[0, 0].tick_params(axis='x', labelsize=labelsize)
ax[0, 0].tick_params(axis='y', labelsize=labelsize)
```

```
# ax[1, 0].plot([0.1, 1], [1, 0], '--', c='k')
for i, (k, v) in enumerate(raps_subgroup_coverage_mean.items()):
    ax[1, 0].plot(ALPHAS, v, label=f'{"missing" if k == 0 else k}', ls=ls,␣
 ↪lw=lw, alpha=alpha, marker='*' if k == 0 else marker, ms=ms, c='gray' if k␣
 ↪== 0 else SKIN_COLORS[k-1])
# ax[1, 0].axvline(x=0.05, color="black", linestyle='--', lw=lw)
# ax[1, 0].text(0.08, 0.11, '95% confidence level', rotation=0,␣
 ↪fontsize=fontsize)
ax[1, 0].set_title('RAPS', fontsize=fontsize)
ax[1, 0].set_xlim(*xlim)
ax[1, 0].set_ylim(*ylim)
ax[1, 0].set_xlabel(r'$\alpha$', fontsize=fontsize)
ax[1, 0].set_ylabel('coverage', fontsize=fontsize)
ax[1, 0].set_xticks(xticks)
ax[1, 0].set_yticks(yticks)
ax[1, 0].tick_params(axis='x', labelsize=labelsize)
ax[1, 0].tick_params(axis='y', labelsize=labelsize)

# ax[0, 1].plot([0, 1], [1, 0], '--', c='k')
for i, (k, v) in enumerate(gaps_subgroup_coverage_mean.items()):
    ax[0, 1].plot(ALPHAS, v, label=f'{"missing" if k == 0 else k}', ls=ls,␣
 ↪lw=lw, alpha=alpha, marker='*' if k == 0 else marker, ms=ms, c='gray' if k␣
 ↪== 0 else SKIN_COLORS[k-1])
# ax[0, 1].axvline(x=0.05, color="black", linestyle='--', lw=lw)
# ax[0, 1].text(0.08, 0.11, '95% confidence level', rotation=0,␣
 ↪fontsize=fontsize)
ax[0, 1].set_title('GAPS', fontsize=fontsize)
ax[0, 1].set_ylim(*ylim)
ax[0, 1].set_ylim(*ylim)
ax[0, 1].set_xlabel(r'$\alpha$', fontsize=fontsize)
ax[0, 1].set_ylabel('coverage', fontsize=fontsize)
ax[0, 1].set_xticks(xticks)
ax[0, 1].set_yticks(yticks)
ax[0, 1].tick_params(axis='x', labelsize=labelsize )
ax[0, 1].tick_params(axis='y', labelsize=labelsize)

# ax[1, 1].plot([0, 1], [1, 0], '--', c='k')
for i, (k, v) in enumerate(graps_subgroup_coverage_mean.items()):
    ax[1, 1].plot(ALPHAS, v, label=f'{"missing" if k == 0 else k}', ls=ls,␣
 ↪lw=lw, alpha=alpha, marker='*' if k == 0 else marker, ms=ms, c='gray' if k␣
 ↪== 0 else SKIN_COLORS[k-1])
# ax[1, 1].axvline(x=0.05, color="black", linestyle='--', lw=lw)
# ax[1, 1].text(0.08, 0.11, '95% confidence level', rotation=0,␣
 ↪fontsize=fontsize)
ax[1, 1].set_title('GRAPS', fontsize=fontsize)
ax[1, 1].set_xlim(*xlim)
```

```
ax[1, 1].set_ylim(*ylim)
ax[1, 1].set_xlabel(r'$\alpha$', fontsize=fontsize)
ax[1, 1].set_ylabel('coverage', fontsize=fontsize)
ax[1, 1].set_xticks(xticks)
ax[1, 1].set_yticks(yticks)
ax[1, 1].tick_params(axis='x', labelsize=labelsize )
ax[1, 1].tick_params(axis='y', labelsize=labelsize)

xticks = np.arange(0.0, 0.6, 0.1)
yticks = np.arange(0, 1.1, 0.1)
xlim = (0, 0.55)
ylim = (1, 90)

for i, (k, v) in enumerate(aps_subgroup_cardinality_mean.items()):
    ax[0, 2].plot(ALPHAS, v, label=f'{"missing" if k == 0 else k}', ls=ls,␣
 ↪lw=lw, alpha=alpha, marker='*' if k == 0 else marker, ms=ms, c='gray' if k␣
 ↪== 0 else SKIN_COLORS[k-1])
ax[0, 2].set_xlim(*xlim)
ax[0, 2].set_ylim(*ylim)
ax[0, 2].set_title('APS', fontsize=fontsize )
ax[0, 2].set_xlabel(r'$\alpha$', fontsize=fontsize)
ax[0, 2].set_ylabel('set size', fontsize=fontsize)
ax[0, 2].set_xticks(xticks)
ax[0, 2].tick_params(axis='x', labelsize=labelsize)
ax[0, 2].tick_params(axis='y', labelsize=labelsize)

for i, (k, v) in enumerate(raps_subgroup_cardinality_mean.items()):
    ax[1, 2].plot(ALPHAS, v, label=f'{"missing" if k == 0 else k}', ls=ls,␣
 ↪lw=lw, alpha=alpha, marker='*' if k == 0 else marker, ms=ms, c='gray' if k␣
 ↪== 0 else SKIN_COLORS[k-1])
ax[1, 2].set_xlim(*xlim)
ax[1, 2].set_ylim(*ylim)
ax[1, 2].set_title('RAPS', fontsize=fontsize )
ax[1, 2].set_xlabel(r'$\alpha$', fontsize=fontsize)
ax[1, 2].set_ylabel('set size', fontsize=fontsize)
ax[1, 2].set_xticks(xticks)
ax[1, 2].tick_params(axis='x', labelsize=labelsize)
ax[1, 2].tick_params(axis='y', labelsize=labelsize)

for i, (k, v) in enumerate(gaps_subgroup_cardinality_mean.items()):
    ax[0, 3].plot(ALPHAS, v, label=f'{"missing" if k == 0 else k}', ls=ls,␣
 ↪lw=lw, alpha=alpha, marker='*' if k == 0 else marker, ms=ms, c='gray' if k␣
 ↪== 0 else SKIN_COLORS[k-1])
ax[0, 3].set_xlim(*xlim)
ax[0, 3].set_ylim(*ylim)
ax[0, 3].set_title('GAPS', fontsize=fontsize )
ax[0, 3].set_xlabel(r'$\alpha$', fontsize=fontsize)
```

```
ax[0, 3].set_ylabel('set size', fontsize=fontsize)
ax[0, 3].set_xticks(xticks)
ax[0, 3].tick_params(axis='x', labelsize=labelsize)
ax[0, 3].tick_params(axis='y', labelsize=labelsize)

for i, (k, v) in enumerate(graps_subgroup_cardinality_mean.items()):
    ax[1, 3].plot(ALPHAS, v, label=f'{"missing" if k == 0 else k}', ls=ls,␣
 ↪lw=lw, alpha=alpha, marker='*' if k == 0 else marker, ms=ms, c='gray' if k␣
 ↪== 0 else SKIN_COLORS[k-1])
ax[1, 3].set_xlim(*xlim)
ax[1, 3].set_ylim(*ylim)
ax[1, 3].set_title('GRAPS', fontsize=fontsize )
ax[1, 3].set_xlabel(r'$\alpha$', fontsize=fontsize)
ax[1, 3].set_ylabel('set size', fontsize=fontsize)
ax[1, 3].set_xticks(xticks)
ax[1, 3].tick_params(axis='x', labelsize=labelsize)
ax[1, 3].tick_params(axis='y', labelsize=labelsize)

handles, labels = ax[0, 0].get_legend_handles_labels()


plt.tight_layout(h_pad=4, w_pad=4)
plt.subplots_adjust(bottom=0.25)
plt.legend(
    handles,
    labels,
    title='fitzpatrick skin type',
    title_fontproperties={
        'style': 'normal',
        'size': fontsize,
    },
    fontsize=fontsize,
    bbox_to_anchor=(0.90, -0.20, 0.0, 0),
    ncol=7,
)

plt.savefig(fig_dir / 'fitz-subgroup-coverage.pdf', bbox_inches="tight")
plt.show()
```

```
'created' timestamp seems very low; regarding as unix timestamp
'modified' timestamp seems very low; regarding as unix timestamp
```



We compare performance between Fitzpatrick skin types in terms of marginal coverage and set size between aggregate conformal methods (APS and RAPS) and their group variants (GAPS and GRAPS) in Figure 5

6.1.14 Measure subgroup coverage / cardinaltiy disparity

[51]:
```
def get_diff(res):
    diff = []
    for a, b in itertools.combinations(list(range(len(res))), 2):
        if a == b: continue
        diff.append(abs(res[a] - res[b]))
    return diff
```

[52]:
```
coverage_disparity = {}
print('COVERAGE'.center(20, '='))
for a in ALPHAS:
    naive_sub_cov = collections.defaultdict(list)
    for k, df in test_dfs.items():
        for sub, cov in get_subgroup_coverage(df, pred_col=f'naive_{a}').
 ↪items():
            naive_sub_cov[sub].append(cov)
    naive_sub_cov = dict(naive_sub_cov)

    aps_sub_cov = collections.defaultdict(list)
    for k, df in test_dfs.items():
        for sub, cov in get_subgroup_coverage(df, pred_col=f'aps_{a}').items():
            aps_sub_cov[sub].append(cov)
    aps_sub_cov = dict(aps_sub_cov)
```

```
    raps_sub_cov = collections.defaultdict(list)
    for k, df in test_dfs.items():
        for sub, cov in get_subgroup_coverage(df, pred_col=f'raps_{a}').items():
            raps_sub_cov[sub].append(cov)
    raps_sub_cov = dict(raps_sub_cov)

    gaps_sub_cov = collections.defaultdict(list)
    for k, df in test_dfs.items():
        for sub, cov in get_subgroup_coverage(df, pred_col=f'gaps_{a}').items():
            gaps_sub_cov[sub].append(cov)
    gaps_sub_cov = dict(gaps_sub_cov)

    graps_sub_cov = collections.defaultdict(list)
    for k, df in test_dfs.items():
        for sub, cov in get_subgroup_coverage(df, pred_col=f'graps_{a}').
↪items():
            graps_sub_cov[sub].append(cov)
    graps_sub_cov = dict(graps_sub_cov)

    print('\nalpha:', a)
    g = lambda x: round(np.mean(get_diff(np.array(list(x.values())).mean(1) - 1␣
↪- a)), 3)
    h = lambda x: round(np.std(get_diff(np.array(list(x.values())).mean(1) - 1␣
↪- a)), 3)
    f = lambda x: (g(x), h(x))
    print('naive\t', f(naive_sub_cov))
    print('aps\t', f(aps_sub_cov))
    print('raps\t', f(raps_sub_cov))
    print('gaps\t', f(gaps_sub_cov))
    print('graps\t', f(graps_sub_cov))

    coverage_disparity[a] = {
        'naive': f(naive_sub_cov),
        'aps': f(aps_sub_cov),
        'raps': f(raps_sub_cov),
        'gaps': f(gaps_sub_cov),
        'graps': f(graps_sub_cov),
    }
```

```
======COVERAGE======

alpha: 0.05
naive    (0.012, 0.007)
aps      (0.009, 0.006)
raps     (0.009, 0.006)
gaps     (0.011, 0.007)
```

```
graps    (0.009, 0.005)

alpha: 0.1
naive    (0.023, 0.015)
aps      (0.015, 0.01)
raps     (0.016, 0.01)
gaps     (0.016, 0.01)
graps    (0.016, 0.01)

alpha: 0.15
naive    (0.03, 0.02)
aps      (0.023, 0.015)
raps     (0.025, 0.015)
gaps     (0.021, 0.013)
graps    (0.019, 0.011)

alpha: 0.2
naive    (0.038, 0.024)
aps      (0.032, 0.02)
raps     (0.032, 0.02)
gaps     (0.02, 0.015)
graps    (0.019, 0.012)

alpha: 0.25
naive    (0.046, 0.032)
aps      (0.036, 0.024)
raps     (0.035, 0.023)
gaps     (0.023, 0.017)
graps    (0.023, 0.016)

alpha: 0.3
naive    (0.051, 0.033)
aps      (0.04, 0.027)
raps     (0.039, 0.026)
gaps     (0.031, 0.02)
graps    (0.028, 0.017)

alpha: 0.35
naive    (0.057, 0.035)
aps      (0.042, 0.028)
raps     (0.039, 0.027)
gaps     (0.033, 0.02)
graps    (0.03, 0.019)

alpha: 0.4
naive    (0.063, 0.038)
aps      (0.046, 0.031)
raps     (0.043, 0.03)
```

```
gaps     (0.033, 0.021)
graps    (0.03, 0.018)

alpha: 0.45
naive    (0.066, 0.04)
aps      (0.047, 0.031)
raps     (0.041, 0.029)
gaps     (0.034, 0.024)
graps    (0.024, 0.017)

alpha: 0.5
naive    (0.071, 0.042)
aps      (0.049, 0.031)
raps     (0.041, 0.025)
gaps     (0.036, 0.022)
graps    (0.029, 0.018)
```

From the table we see lower subgroup coverage disparity (the first number in parenthesis) for GAPS and GRAPS

[53]:
```
cardinality_disparity = {}
print('CARDINALITY'.center(20, '='))
for a in ALPHAS:
    naive_sub_card = collections.defaultdict(list)
    for k, df in test_dfs.items():
        for sub, card in get_subgroup_cardinality(df, pred_col=f'naive_{a}').
 ↪items():
            naive_sub_card[sub].append(card)
    naive_sub_card = dict(naive_sub_card)

    aps_sub_card = collections.defaultdict(list)
    for k, df in test_dfs.items():
        for sub, card in get_subgroup_cardinality(df, pred_col=f'aps_{a}').
 ↪items():
            aps_sub_card[sub].append(card)
    aps_sub_card = dict(aps_sub_card)

    raps_sub_card = collections.defaultdict(list)
    for k, df in test_dfs.items():
        for sub, card in get_subgroup_cardinality(df, pred_col=f'raps_{a}').
 ↪items():
            raps_sub_card[sub].append(card)
    raps_sub_card = dict(raps_sub_card)

    gaps_sub_card = collections.defaultdict(list)
    for k, df in test_dfs.items():
        for sub, card in get_subgroup_cardinality(df, pred_col=f'gaps_{a}').
 ↪items():
```

```
            gaps_sub_card[sub].append(card)
    gaps_sub_card = dict(gaps_sub_card)

    graps_sub_card = collections.defaultdict(list)
    for k, df in test_dfs.items():
        for sub, card in get_subgroup_cardinality(df, pred_col=f'graps_{a}').
↪items():
            graps_sub_card[sub].append(card)
    graps_sub_card = dict(graps_sub_card)

    print('\nalpha:', a)
    g = lambda x: round(np.mean(get_diff(np.array(list(x.values())).mean(1) - 1␣
↪- a)), 1)
    h = lambda x: round(np.std(get_diff(np.array(list(x.values())).mean(1) - 1␣
↪- a)), 1)
    f = lambda x: (g(x), h(x))
    print('naive\t', f(naive_sub_card))
    print('aps\t', f(aps_sub_card))
    print('raps\t', f(raps_sub_card))
    print('gaps\t', f(gaps_sub_card))
    print('graps\t', f(graps_sub_card))

    cardinality_disparity[a] = {
        'naive': f(naive_sub_card),
        'aps': f(aps_sub_card),
        'raps': f(raps_sub_card),
        'gaps': f(gaps_sub_card),
        'graps': f(graps_sub_card),
    }
```

```
====CARDINALITY=====

alpha: 0.05
naive    (3.4, 2.0)
aps      (3.1, 1.7)
raps     (2.9, 1.7)
gaps     (9.4, 5.9)
graps    (5.8, 3.4)

alpha: 0.1
naive    (2.8, 1.7)
aps      (3.1, 1.8)
raps     (3.0, 1.7)
gaps     (8.1, 5.2)
graps    (6.4, 4.0)

alpha: 0.15
```

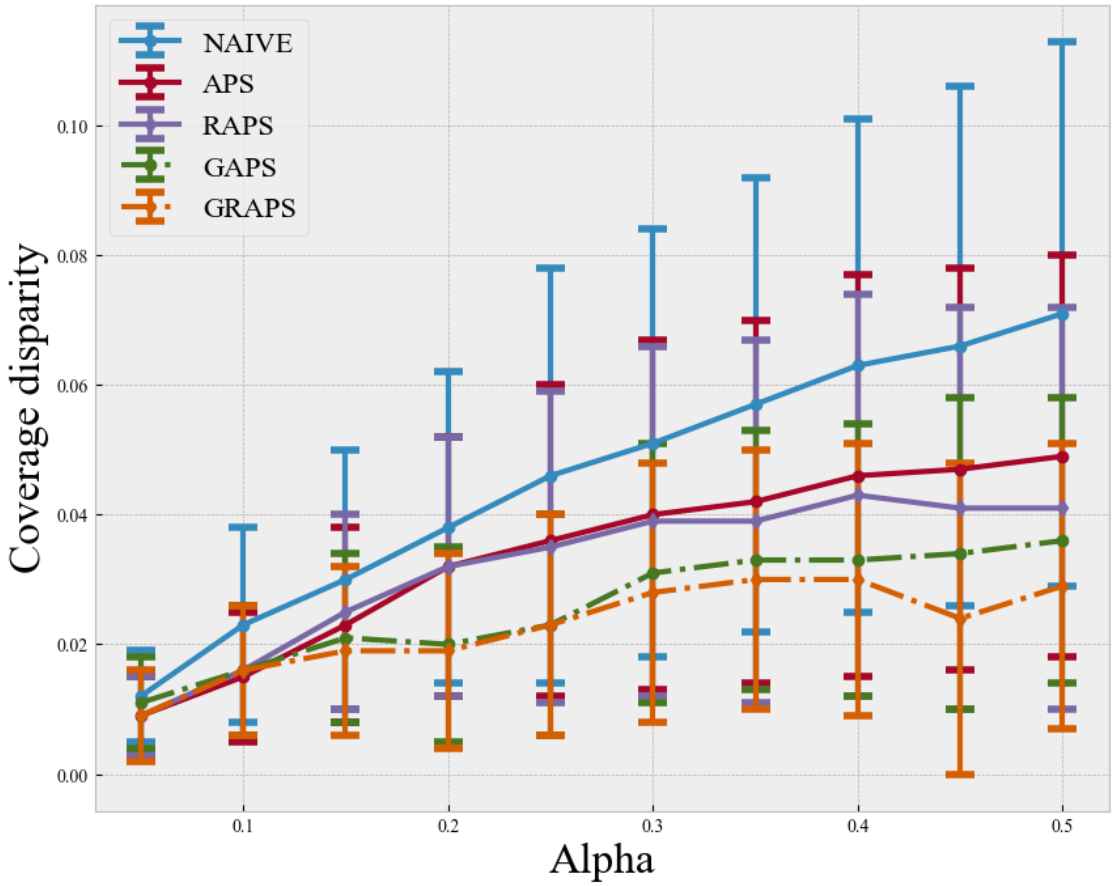
```
naive    (2.2, 1.4)
aps      (2.9, 1.7)
raps     (2.8, 1.6)
gaps     (8.4, 5.0)
graps    (7.6, 4.6)

alpha: 0.2
naive    (1.7, 1.1)
aps      (2.6, 1.5)
raps     (2.5, 1.5)
gaps     (6.4, 3.7)
graps    (4.9, 2.9)

alpha: 0.25
naive    (1.3, 0.8)
aps      (2.3, 1.4)
raps     (2.3, 1.4)
gaps     (5.3, 3.0)
graps    (4.5, 2.7)

alpha: 0.3
naive    (1.0, 0.6)
aps      (2.1, 1.3)
raps     (2.0, 1.2)
gaps     (4.2, 2.4)
graps    (3.9, 2.3)

alpha: 0.35
naive    (0.6, 0.4)
aps      (1.8, 1.1)
raps     (1.7, 1.1)
gaps     (3.5, 2.0)
graps    (3.3, 2.0)

alpha: 0.4
naive    (0.4, 0.3)
aps      (1.6, 1.0)
raps     (1.5, 1.0)
gaps     (2.9, 1.8)
graps    (2.5, 1.5)

alpha: 0.45
naive    (0.3, 0.2)
aps      (1.4, 0.9)
raps     (1.3, 0.9)
gaps     (2.5, 1.6)
graps    (2.2, 1.5)
```

```
alpha: 0.5
naive    (0.2, 0.1)
aps      (1.2, 0.8)
raps     (1.1, 0.7)
gaps     (2.0, 1.3)
graps    (1.7, 1.1)
```

Higher subgroup disparity for GAPS and GRAPS

[54]:
```
plt.figure(figsize=(10, 8))

plt.xticks(np.arange(0, 1.1, 0.1))
plt.xlabel('Alpha', fontsize=24)
plt.ylabel('Coverage disparity', fontsize=24)

plt.errorbar(
    coverage_disparity.keys(),
    [x['naive'][0] for x in coverage_disparity.values()],
    yerr=[x['naive'][1] for x in coverage_disparity.values()],
    marker='o',
    lw=3,
    capsize=8,
    capthick=4,
    label='NAIVE',
)
plt.errorbar(
    coverage_disparity.keys(),
    [x['aps'][0] for x in coverage_disparity.values()],
    yerr=[x['aps'][1] for x in coverage_disparity.values()],
    marker='o',
    lw=3,
    capsize=8,
    capthick=4,
    label='APS',
)
plt.errorbar(
    coverage_disparity.keys(),
    [x['raps'][0] for x in coverage_disparity.values()],
    yerr=[x['aps'][1] for x in coverage_disparity.values()],
    marker='d',
    lw=3,
    capsize=8,
    capthick=4,
    label='RAPS',
)
plt.errorbar(
    coverage_disparity.keys(),
    [x['gaps'][0] for x in coverage_disparity.values()],
```

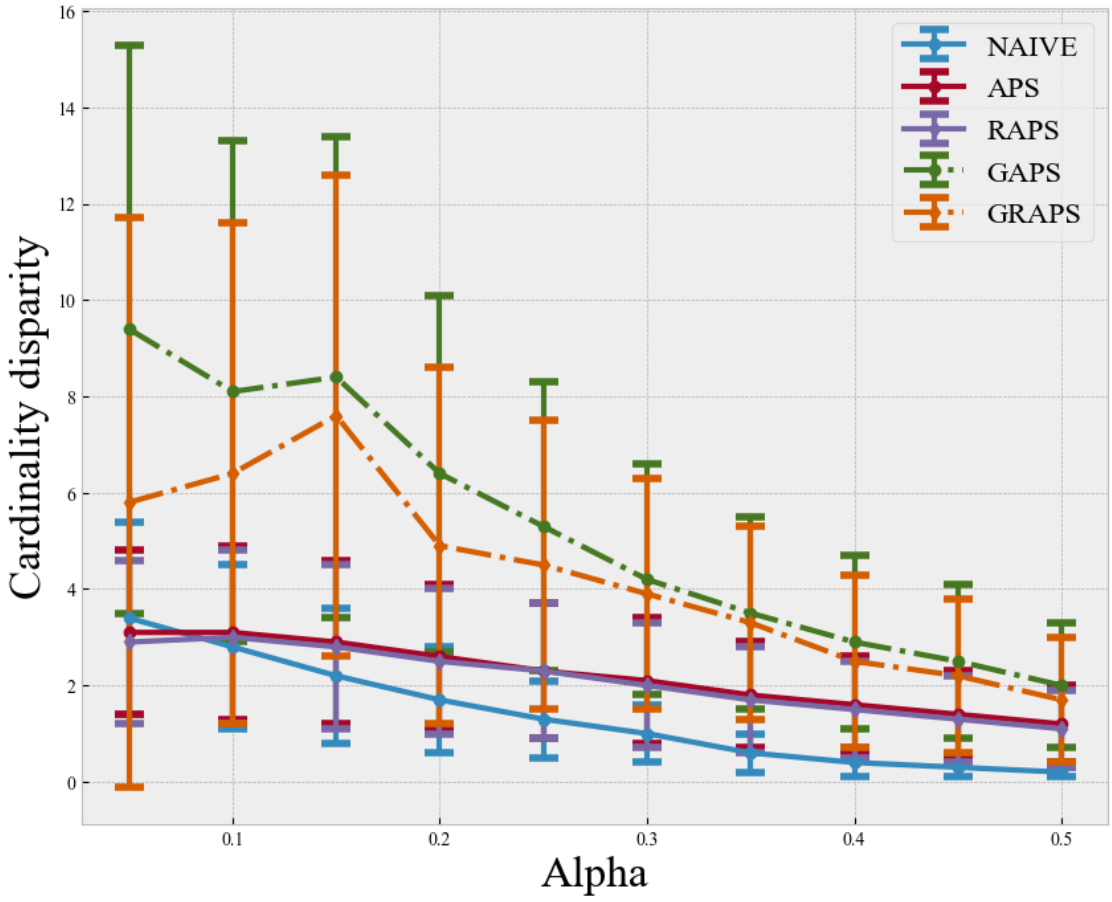
```
    yerr=[x['gaps'][1] for x in coverage_disparity.values()],
    marker='o',
    ls='-.',
    lw=3,
    capsize=8,
    capthick=4,
    label='GAPS',
)
plt.errorbar(
    coverage_disparity.keys(),
    [x['graps'][0] for x in coverage_disparity.values()],
    yerr=[x['gaps'][1] for x in coverage_disparity.values()],
    marker='d',
    ls='-.',
    lw=3,
    capsize=8,
    capthick=4,
    label='GRAPS',
)
plt.legend(fontsize=16)
plt.savefig(fig_dir / 'fitz-coverage-disparity.png')
plt.show()
```



Same thing ploted as graph

[55]:
```
plt.figure(figsize=(10, 8))

plt.xticks(np.arange(0, 1.1, 0.1))
plt.xlabel('Alpha', fontsize=24)
plt.ylabel('Cardinality disparity', fontsize=24)

plt.errorbar(
    cardinality_disparity.keys(),
    [x['naive'][0] for x in cardinality_disparity.values()],
    yerr=[x['naive'][1] for x in cardinality_disparity.values()],
    marker='o',
    lw=3,
    capsize=8,
    capthick=4,
    label='NAIVE',
)
plt.errorbar(
```

```
    cardinality_disparity.keys(),
    [x['aps'][0] for x in cardinality_disparity.values()],
    yerr=[x['aps'][1] for x in cardinality_disparity.values()],
    marker='o',
    lw=3,
    capsize=8,
    capthick=4,
    label='APS',
)
plt.errorbar(
    cardinality_disparity.keys(),
    [x['raps'][0] for x in cardinality_disparity.values()],
    yerr=[x['aps'][1] for x in cardinality_disparity.values()],
    marker='d',
    lw=3,
    capsize=8,
    capthick=4,
    label='RAPS',
)
plt.errorbar(
    cardinality_disparity.keys(),
    [x['gaps'][0] for x in cardinality_disparity.values()],
    yerr=[x['gaps'][1] for x in cardinality_disparity.values()],
    marker='o',
    ls='-.',
    lw=3,
    capsize=8,
    capthick=4,
    label='GAPS',
)
plt.errorbar(
    cardinality_disparity.keys(),
    [x['graps'][0] for x in cardinality_disparity.values()],
    yerr=[x['gaps'][1] for x in cardinality_disparity.values()],
    marker='d',
    ls='-.',
    lw=3,
    capsize=8,
    capthick=4,
    label='GRAPS',
)
plt.legend(fontsize=16)
plt.savefig(fig_dir / 'fitz-cardinality-disparity.png')
plt.show()
```



[56]:
```
!pip install matplotlib==3.5.1
```

```
Requirement already satisfied: matplotlib==3.5.1 in
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages (3.5.1)
Requirement already satisfied: cycler>=0.10 in
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages (from
matplotlib==3.5.1) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages (from
matplotlib==3.5.1) (4.49.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages (from
matplotlib==3.5.1) (1.4.5)
Requirement already satisfied: numpy>=1.17 in
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages (from
matplotlib==3.5.1) (1.22.2)
Requirement already satisfied: packaging>=20.0 in
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages (from
matplotlib==3.5.1) (23.2)
```

```
Requirement already satisfied: pillow>=6.2.0 in
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages (from
matplotlib==3.5.1) (9.0.0)
Requirement already satisfied: pyparsing>=2.2.1 in
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages (from
matplotlib==3.5.1) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages (from
matplotlib==3.5.1) (2.9.0)
Requirement already satisfied: six>=1.5 in
/opt/anaconda3/envs/dlh38env/lib/python3.8/site-packages (from python-
dateutil>=2.7->matplotlib==3.5.1) (1.16.0)
```

6.1.15 Distribution of prediction set sizes by subgroup

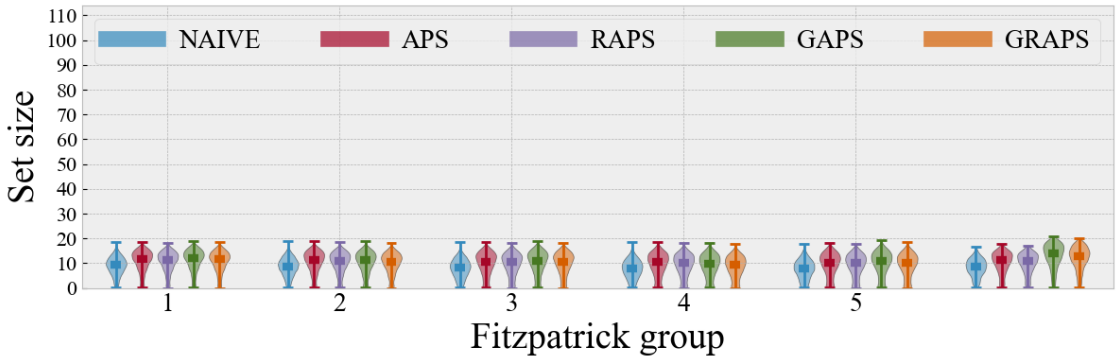
[57]:
```
fontsize = 24
width = 0.12
alpha = 0.1
# alpha = 0.2
temp_var = None

def set_box_color(bp, color):
    plt.setp(bp['boxes'], facecolor=color)
    plt.setp(bp['whiskers'], color=color)
    plt.setp(bp['caps'], color=color)
    plt.setp(bp['medians'], color='k')

def set_violin_color(violin, color):
    for pc in violin['bodies']:
        pc.set_facecolor(color)
        pc.set_edgecolor('black')
        pc.set_alpha(0.5)

    violin['cbars'].set_edgecolor(color)
    violin['cmaxes'].set_edgecolor(color)
    violin['cmins'].set_edgecolor(color)
    violin['cmedians'].set_edgecolor(color)
    violin['cmedians'].set_linewidth(6)

plt.figure(figsize=(12, 4))
plt.xticks(A[1:], fontsize=fontsize - 4)
plt.xlabel('Fitzpatrick group', fontsize=fontsize + 4)
plt.ylabel('Set size', fontsize=fontsize + 4)
plt.yticks(range(0, 120, 10), fontsize=fontsize - 8)
plt.xlim(0.5, 6.5)
plt.ylim(0, 114)
```

```
for i in range(1, len(A)):
    violin = plt.violinplot(
        np.array([
            df[df.subgroup == i].apply(lambda x:␣
 ↪len(x[f'naive_{alpha}']),axis=1).tolist()
            for df in test_dfs.values()
        ]).mean(0),
        positions=[i - 0.30],
        widths=width,
        showmedians=True,
    )
    set_violin_color(violin, 'C0')


for i in range(1, len(A)):
    violin = plt.violinplot(
        np.array([
            df[df.subgroup == i].apply(lambda x: len(x[f'aps_{alpha}']),␣
 ↪axis=1).tolist()
            for df in test_dfs.values()
        ]).mean(0),
        positions=[i - 0.15],
        widths=width,
        showmedians=True,
    )
    set_violin_color(violin, 'C1')

for i in range(1, len(A)):
    violin = plt.violinplot(
        np.array([
            df[df.subgroup == i].apply(lambda x: len(x[f'raps_{alpha}']),␣
 ↪axis=1).tolist()
            for df in test_dfs.values()
        ]).mean(0),
        showmedians=True,
        positions=[i + 0.00],
        widths=width,
    )
    set_violin_color(violin, 'C2')

for i in range(1, len(A)):
    violin = plt.violinplot(
        np.array([
            df[df.subgroup == i].apply(lambda x: len(x[f'gaps_{alpha}']),␣
 ↪axis=1).tolist()
            for df in test_dfs.values()
        ]).mean(0),
```

```
        positions=[i+0.15],
        widths=width,
        showmedians=True,

    )
    set_violin_color(violin, 'C3')


for i in range(1, len(A)):
    violin = plt.violinplot(
        np.array([
            df[df.subgroup == i].apply(lambda x: len(x[f'graps_{alpha}']),␣
 ↪axis=1).tolist()
            for df in test_dfs.values()
        ]).mean(0),
        positions=[i+0.30],
        widths=width,
        showmedians=True,
    )
    set_violin_color(violin, 'C4')


a, = plt.plot([0,0], [0, 0],'C0', lw=12, alpha=0.7)
b, = plt.plot([0,0], [0, 0],'C1', lw=12, alpha=0.7)
c, = plt.plot([0,0], [0,0],'C2', lw=12, alpha=0.7)
d, = plt.plot([0,0], [0, 0],'C3', lw=12, alpha=0.7)
e, = plt.plot([0,0], [0, 0],'C4', lw=12, alpha=0.7)

plt.legend(
    (a, b, c, d, e),
    ['NAIVE', 'APS', 'RAPS', 'GAPS', 'GRAPS'],
    fontsize=fontsize - 4,
    ncol=5,
    mode='expand',
    loc='upper center'
)
plt.tight_layout()
plt.savefig(fig_dir / 'fitz-boxplot-set-sizes.png')
plt.show()
```

```
/var/folders/xz/h7s8jn114b9bj0bkdj268bwc0000gn/T/ipykernel_65293/2088735395.py:3
5: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  np.array([
/var/folders/xz/h7s8jn114b9bj0bkdj268bwc0000gn/T/ipykernel_65293/2088735395.py:4
8: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
```

```
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  np.array([
/var/folders/xz/h7s8jn114b9bj0bkdj268bwc0000gn/T/ipykernel_65293/2088735395.py:6
0: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  np.array([
/var/folders/xz/h7s8jn114b9bj0bkdj268bwc0000gn/T/ipykernel_65293/2088735395.py:7
2: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  np.array([
/var/folders/xz/h7s8jn114b9bj0bkdj268bwc0000gn/T/ipykernel_65293/2088735395.py:8
6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  np.array([
```



6.1.16 Ruling in and ruling out critical conditions

[58]:
```
malignant_conditions = {
    reversed_label_map[x]: x for x in
    skin_df.query('three_partition_label == "malignant"').raw_label.unique().
  ↪tolist()
}
malignant_conditions
```

[58]:
```
{42: 'kaposi sarcoma',
 56: 'melanoma',
```

```
 5: 'actinic keratosis',
 8: 'basal cell carcinoma',
 9: 'basal cell carcinoma morpheiform',
 60: 'mycosis fungoides',
 103: 'superficial spreading melanoma ssm',
 98: 'squamous cell carcinoma',
 46: 'lentigo maligna',
 97: 'solid cystic basal cell carcinoma',
 55: 'malignant melanoma'}
```

[59]:
```python
def rule_in(df, pred='aps_0.05', malignant_conditions=malignant_conditions.
↪keys()):
    correct = 0
    total = 0

    for i, row in df.iterrows():
        if row['label'] in malignant_conditions:
            total += 1
            if row['label'] in row[pred]:
                correct += 1

    return correct / total

def rule_out(df, pred='aps_0.05'):
    correct = 0
    total = 0
    for i, row in df.iterrows():
        if row['label'] not in malignant_conditions:
            total += 1
            if len(set(row[pred]) & set(malignant_conditions)) == 0:
                correct += 1

    return correct / total
```

[60]:
```python
naive_rule_in = collections.defaultdict(list)
aps_rule_in = collections.defaultdict(list)
gaps_rule_in = collections.defaultdict(list)
raps_rule_in = collections.defaultdict(list)
graps_rule_in = collections.defaultdict(list)

naive_rule_out = collections.defaultdict(list)
aps_rule_out = collections.defaultdict(list)
gaps_rule_out = collections.defaultdict(list)
raps_rule_out = collections.defaultdict(list)
graps_rule_out = collections.defaultdict(list)

naive_crit_set_size = collections.defaultdict(list)
```

```
aps_crit_set_size = collections.defaultdict(list)
gaps_crit_set_size = collections.defaultdict(list)
raps_crit_set_size = collections.defaultdict(list)
graps_crit_set_size = collections.defaultdict(list)

naive_non_crit_set_size = collections.defaultdict(list)
aps_non_crit_set_size = collections.defaultdict(list)
gaps_non_crit_set_size = collections.defaultdict(list)
raps_non_crit_set_size = collections.defaultdict(list)
graps_non_crit_set_size = collections.defaultdict(list)

for alpha in ALPHAS:
    for seed in range(RUNS):
        df = test_dfs[f'seed_{seed}']
        crit_df = df[df.label.isin(malignant_conditions)]
        non_crit_df = df[~df.label.isin(malignant_conditions)]

        naive_rule_in[alpha].append(rule_in(crit_df, f'naive_{alpha}'))
        aps_rule_in[alpha].append(rule_in(crit_df, f'aps_{alpha}'))
        gaps_rule_in[alpha].append(rule_in(crit_df, f'gaps_{alpha}'))
        raps_rule_in[alpha].append(rule_in(crit_df, f'raps_{alpha}'))
        graps_rule_in[alpha].append(rule_in(crit_df, f'graps_{alpha}'))

        naive_crit_set_size[alpha].append(crit_df[f'naive_{alpha}'].apply(len).
 ↪mean())
        aps_crit_set_size[alpha].append(crit_df[f'aps_{alpha}'].apply(len).
 ↪mean())
        gaps_crit_set_size[alpha].append(crit_df[f'gaps_{alpha}'].apply(len).
 ↪mean())
        raps_crit_set_size[alpha].append(crit_df[f'raps_{alpha}'].apply(len).
 ↪mean())
        graps_crit_set_size[alpha].append(crit_df[f'graps_{alpha}'].apply(len).
 ↪mean())

        naive_rule_out[alpha].append(rule_out(non_crit_df, f'naive_{alpha}'))
        aps_rule_out[alpha].append(rule_out(non_crit_df, f'aps_{alpha}'))
        gaps_rule_out[alpha].append(rule_out(non_crit_df, f'gaps_{alpha}'))
        raps_rule_out[alpha].append(rule_out(non_crit_df, f'raps_{alpha}'))
        graps_rule_out[alpha].append(rule_out(non_crit_df, f'graps_{alpha}'))

        naive_non_crit_set_size[alpha].append(non_crit_df[f'naive_{alpha}'].
 ↪apply(len).mean())
        aps_non_crit_set_size[alpha].append(non_crit_df[f'aps_{alpha}'].
 ↪apply(len).mean())
        gaps_non_crit_set_size[alpha].append(non_crit_df[f'gaps_{alpha}'].
 ↪apply(len).mean())
```

```
        raps_non_crit_set_size[alpha].append(non_crit_df[f'raps_{alpha}'].
 ↪apply(len).mean())
        graps_non_crit_set_size[alpha].append(non_crit_df[f'graps_{alpha}'].
 ↪apply(len).mean())
```

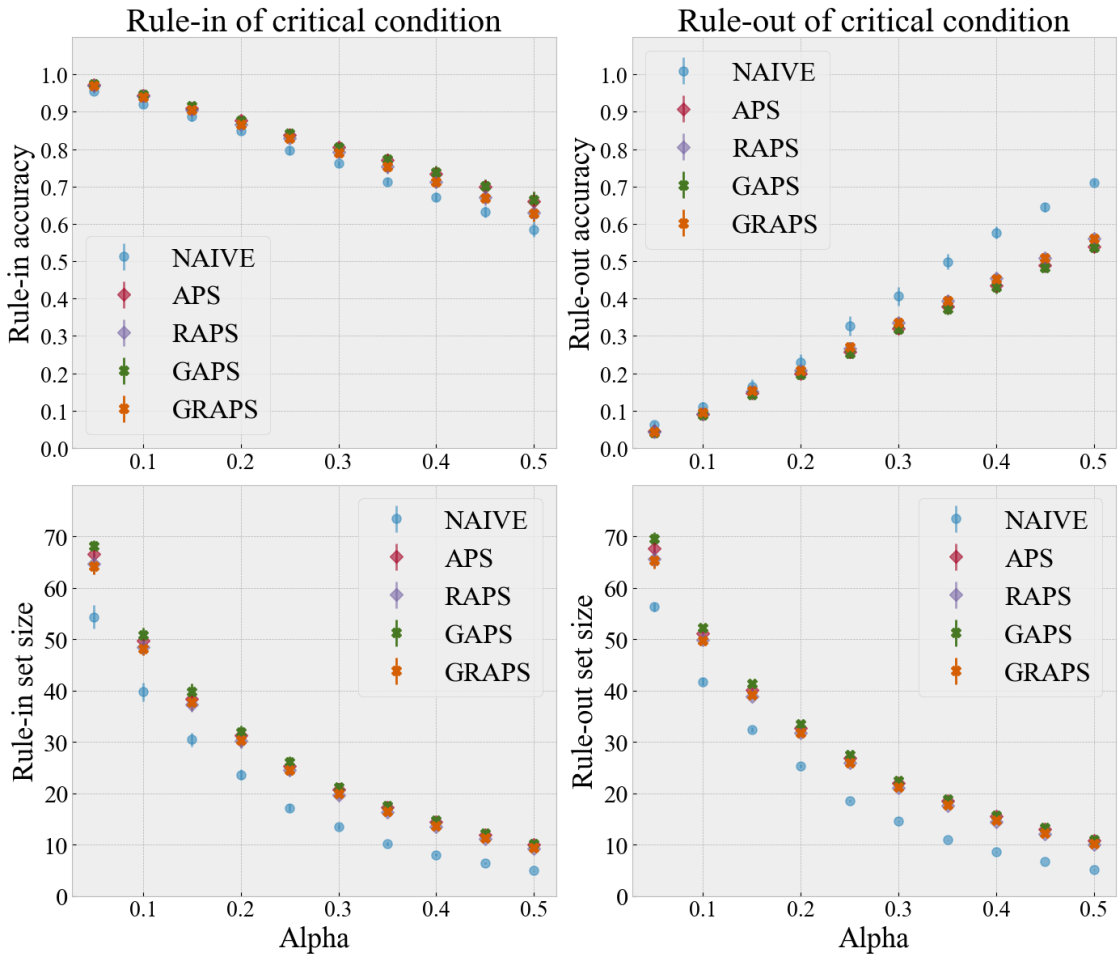
[61]:
```
markersize=8
fontsize=26
alpha=0.8
fmt1='o'
fmt2='X'
fmt3='D'

fig, ax = plt.subplots(2, 2)
fig.set_size_inches(14, 12)

ax[0, 0].set_title('Rule-in of critical condition', fontsize=fontsize + 4)
# ax[0, 0].set_xlabel('Alpha', fontsize=fontsize)
ax[0, 0].set_ylabel('Rule-in accuracy', fontsize=fontsize)
ax[0, 0].set_xticks(np.arange(0, 1.1, 0.1), fontsize=fontsize)
ax[0, 0].set_yticks(np.arange(0, 1.1, 0.1), fontsize=fontsize)
ax[0, 0].set_xticklabels([round(x, 1) for x in np.arange(0, 1.1, 0.1)],␣
 ↪fontsize=fontsize-6)
ax[0, 0].set_yticklabels([round(y, 1) for y in np.arange(0, 1.1, 0.1)],␣
 ↪fontsize=fontsize-6)
ax[0, 0].set_ylim(0, 1.1)

ax[0, 1].set_title('Rule-out of critical condition', fontsize=fontsize + 4)
# ax[0, 1].set_xlabel('Alpha', fontsize=fontsize)
ax[0, 1].set_ylabel('Rule-out accuracy', fontsize=fontsize)
ax[0, 1].set_xticks(np.arange(0, 1.1, 0.1), fontsize=fontsize)
ax[0, 1].set_yticks(np.arange(0, 1.1, 0.1), fontsize=fontsize)
ax[0, 1].set_xticklabels([round(x, 1) for x in np.arange(0, 1.1, 0.1)],␣
 ↪fontsize=fontsize-6)
ax[0, 1].set_yticklabels([round(y, 1) for y in np.arange(0, 1.1, 0.1)],␣
 ↪fontsize=fontsize-6)
ax[0, 1].set_ylim(0, 1.1)

# ax[1, 0].set_title('Rule-in', fontsize=fontsize + 4)
ax[1, 0].set_xlabel('Alpha', fontsize=fontsize)
ax[1, 0].set_ylabel('Rule-in set size', fontsize=fontsize)
ax[1, 0].set_xticks(np.arange(0, 1.1, 0.1), fontsize=fontsize)
ax[1, 0].set_yticks(range(0, 80, 10), fontsize=fontsize)
ax[1, 0].set_xticklabels([round(x, 1) for x in np.arange(0, 1.1, 0.1)],␣
 ↪fontsize=fontsize-6)
ax[1, 0].set_yticklabels(range(0, 80, 10), fontsize=fontsize-4)
ax[1, 0].set_ylim(0, 80)
```

```
# ax[1, 1].set_title('Rule-out', fontsize=fontsize + 4)
ax[1, 1].set_xlabel('Alpha', fontsize=fontsize)
ax[1, 1].set_ylabel('Rule-out set size', fontsize=fontsize)
ax[1, 1].set_xticks(np.arange(0, 1.1, 0.1), fontsize=fontsize)
ax[1, 1].set_yticks(range(0, 80, 10), fontsize=fontsize)
ax[1, 1].set_xticklabels([round(x, 1) for x in np.arange(0, 1.1, 0.1)],␣
 ↪fontsize=fontsize-6)
ax[1, 1].set_yticklabels(range(0, 80, 10), fontsize=fontsize-4)
ax[1, 1].set_ylim(0, 80)

# ax[0, 0].axvline(x=0.05, color="black", linestyle='--', lw=lw)
# ax[0, 1].axvline(x=0.05, color="black", linestyle='--', lw=lw)
# ax[1, 0].axvline(x=0.05, color="black", linestyle='--', lw=lw)
# ax[1, 1].axvline(x=0.05, color="black", linestyle='--', lw=lw)
# ax[0, 0].text(0.1, 0.11, '95% confidence level', rotation=0,␣
 ↪fontsize=fontsize)
# ax[0, 1].text(0.1, 0.11, '95% confidence level', rotation=0,␣
 ↪fontsize=fontsize)
# ax[1, 0].text(0.1, 65, '95% confidence level', rotation=0, fontsize=fontsize)
# ax[1, 1].text(0.1, 65, '95% confidence level', rotation=0, fontsize=fontsize)

naive = np.array(list(naive_rule_in.values()))
ax[0, 0].errorbar(
    ALPHAS, naive.mean(1), naive.std(1),
    fmt=fmt1, color='C0', markersize=markersize,
    label='NAIVE', alpha=alpha - 0.2,
)
aps = np.array(list(aps_rule_in.values()))
ax[0, 0].errorbar(
    ALPHAS, aps.mean(1), aps.std(1),
    fmt=fmt3, color='C1', markersize=markersize,
    label='APS', alpha=alpha - 0.2,
)
raps = np.array(list(raps_rule_in.values()))
ax[0, 0].errorbar(
    ALPHAS, raps.mean(1), raps.std(1),
    fmt=fmt3, color='C2', markersize=markersize,
    label='RAPS', alpha=alpha - 0.2,
)
gaps = np.array(list(gaps_rule_in.values()))
ax[0, 0].errorbar(
    ALPHAS, gaps.mean(1), gaps.std(1),
    fmt=fmt2, color='C3', markersize=markersize,
    label='GAPS', alpha=alpha + 0.2,
)
graps = np.array(list(graps_rule_in.values()))
ax[0, 0].errorbar(
```

```
    ALPHAS, graps.mean(1), graps.std(1),
    fmt=fmt2, color='C4', markersize=markersize,
    label='GRAPS', alpha=alpha + 0.2,
)
ax[0, 0].legend(fontsize=fontsize-2, loc='lower left')

naive = np.array(list(naive_rule_out.values()))
ax[0, 1].errorbar(
    ALPHAS, naive.mean(1), naive.std(1),
    fmt=fmt1, color='C0', markersize=markersize,
    label='NAIVE', alpha=alpha - 0.2,
)
aps = np.array(list(aps_rule_out.values()))
ax[0, 1].errorbar(
    ALPHAS, aps.mean(1), aps.std(1),
    fmt=fmt3, color='C1', markersize=markersize,
    label='APS', alpha=alpha - 0.2,
)
raps = np.array(list(raps_rule_out.values()))
ax[0, 1].errorbar(
    ALPHAS, raps.mean(1), raps.std(1),
    fmt=fmt3, color='C2', markersize=markersize,
    label='RAPS', alpha=alpha - 0.2,
)
gaps = np.array(list(gaps_rule_out.values()))
ax[0, 1].errorbar(
    ALPHAS, gaps.mean(1), gaps.std(1),
    fmt=fmt2, color='C3', markersize=markersize,
    label='GAPS', alpha=alpha + 0.2,
)
graps = np.array(list(graps_rule_out.values()))
ax[0, 1].errorbar(
    ALPHAS, graps.mean(1), graps.std(1),
    fmt=fmt2, color='C4', markersize=markersize,
    label='GRAPS', alpha=alpha + 0.2,
)
ax[0, 1].legend(fontsize=fontsize-2, loc='upper left')

naive = np.array(list(naive_crit_set_size.values()))
ax[1, 0].errorbar(
    ALPHAS, naive.mean(1), naive.std(1),
    fmt=fmt1, color='C0', markersize=markersize,
    label='NAIVE', alpha=alpha - 0.2,
)
aps = np.array(list(aps_crit_set_size.values()))
ax[1, 0].errorbar(
    ALPHAS, aps.mean(1), aps.std(1),
```

```
    fmt=fmt3, color='C1', markersize=markersize,
    label='APS', alpha=alpha - 0.2,
)
raps = np.array(list(raps_crit_set_size.values()))
ax[1, 0].errorbar(
    ALPHAS, raps.mean(1), raps.std(1),
    fmt=fmt3, color='C2', markersize=markersize,
    label='RAPS', alpha=alpha - 0.2,
)
gaps = np.array(list(gaps_crit_set_size.values()))
ax[1, 0].errorbar(
    ALPHAS, gaps.mean(1), gaps.std(1),
    fmt=fmt2, color='C3', markersize=markersize,
    label='GAPS', alpha=alpha + 0.2,
)
graps = np.array(list(graps_crit_set_size.values()))
ax[1, 0].errorbar(
    ALPHAS, graps.mean(1), graps.std(1),
    fmt=fmt2, color='C4', markersize=markersize,
    label='GRAPS', alpha=alpha + 0.2,
)
ax[1, 0].legend(fontsize=fontsize-2, loc='upper right')

naive = np.array(list(naive_non_crit_set_size.values()))
ax[1, 1].errorbar(
    ALPHAS, naive.mean(1), naive.std(1),
    fmt=fmt1, color='C0', markersize=markersize,
    label='NAIVE', alpha=alpha - 0.2,
)
aps = np.array(list(aps_non_crit_set_size.values()))
ax[1, 1].errorbar(
    ALPHAS, aps.mean(1), aps.std(1),
    fmt=fmt3, color='C1', markersize=markersize,
    label='APS', alpha=alpha - 0.2,
)
raps = np.array(list(raps_non_crit_set_size.values()))
ax[1, 1].errorbar(
    ALPHAS, raps.mean(1), raps.std(1),
    fmt=fmt3, color='C2', markersize=markersize,
    label='RAPS', alpha=alpha - 0.2,
)
gaps = np.array(list(gaps_non_crit_set_size.values()))
ax[1, 1].errorbar(
    ALPHAS, gaps.mean(1), gaps.std(1),
    fmt=fmt2, color='C3', markersize=markersize,
    label='GAPS', alpha=alpha + 0.2,
)
```

```
graps = np.array(list(graps_non_crit_set_size.values()))
ax[1, 1].errorbar(
    ALPHAS, graps.mean(1), graps.std(1),
    fmt=fmt2, color='C4', markersize=markersize,
    label='GRAPS', alpha=alpha + 0.2,
)
ax[1, 1].legend(fontsize=fontsize -2, loc='upper right')

plt.tight_layout()
plt.savefig(fig_dir / 'fitz-use-case-perf')
plt.show()
```



6.1.17 Comparing rule-in/out between subgroups at $\alpha = 0.1$

[62]:
```
all_group_naive_rule_in = {}
all_group_aps_rule_in = {}
all_group_gaps_rule_in = {}
all_group_raps_rule_in = {}
all_group_graps_rule_in = {}

all_group_naive_rule_out = {}
all_group_aps_rule_out = {}
all_group_gaps_rule_out = {}
all_group_raps_rule_out = {}
all_group_graps_rule_out = {}

all_group_naive_crit_set_size = {}
all_group_aps_crit_set_size = {}
all_group_gaps_crit_set_size = {}
all_group_raps_crit_set_size = {}
all_group_graps_crit_set_size = {}

all_group_naive_non_crit_set_size = {}
all_group_aps_non_crit_set_size = {}
all_group_gaps_non_crit_set_size = {}
all_group_raps_non_crit_set_size = {}
all_group_graps_non_crit_set_size = {}

for a in A:
    group_naive_rule_in = collections.defaultdict(list)
    group_aps_rule_in = collections.defaultdict(list)
    group_gaps_rule_in = collections.defaultdict(list)
    group_raps_rule_in = collections.defaultdict(list)
    group_graps_rule_in = collections.defaultdict(list)

    group_naive_rule_out = collections.defaultdict(list)
    group_aps_rule_out = collections.defaultdict(list)
    group_gaps_rule_out = collections.defaultdict(list)
    group_raps_rule_out = collections.defaultdict(list)
    group_graps_rule_out = collections.defaultdict(list)

    group_naive_crit_set_size = collections.defaultdict(list)
    group_aps_crit_set_size = collections.defaultdict(list)
    group_gaps_crit_set_size = collections.defaultdict(list)
    group_raps_crit_set_size = collections.defaultdict(list)
    group_graps_crit_set_size = collections.defaultdict(list)

    group_naive_non_crit_set_size = collections.defaultdict(list)
    group_aps_non_crit_set_size = collections.defaultdict(list)
```

```
    group_gaps_non_crit_set_size = collections.defaultdict(list)
    group_raps_non_crit_set_size = collections.defaultdict(list)
    group_graps_non_crit_set_size = collections.defaultdict(list)

    for alpha in ALPHAS:
        for seed in range(RUNS):
            df = test_dfs[f'seed_{seed}']
            sub_df = df.query('subgroup == @a')
            crit_df = sub_df[sub_df.label.isin(malignant_conditions)]
            non_crit_df = sub_df[~sub_df.label.isin(malignant_conditions)]

            group_naive_rule_in[alpha].append(rule_in(crit_df,␣
↪f'naive_{alpha}'))
            group_aps_rule_in[alpha].append(rule_in(crit_df, f'aps_{alpha}'))
            group_gaps_rule_in[alpha].append(rule_in(crit_df, f'gaps_{alpha}'))
            group_raps_rule_in[alpha].append(rule_in(crit_df, f'raps_{alpha}'))
            group_graps_rule_in[alpha].append(rule_in(crit_df,␣
↪f'graps_{alpha}'))

            group_naive_crit_set_size[alpha].append(crit_df[f'naive_{alpha}'].
↪apply(len).mean())
            group_aps_crit_set_size[alpha].append(crit_df[f'aps_{alpha}'].
↪apply(len).mean())
            group_gaps_crit_set_size[alpha].append(crit_df[f'gaps_{alpha}'].
↪apply(len).mean())
            group_raps_crit_set_size[alpha].append(crit_df[f'raps_{alpha}'].
↪apply(len).mean())
            group_graps_crit_set_size[alpha].append(crit_df[f'graps_{alpha}'].
↪apply(len).mean())

            group_naive_rule_out[alpha].append(rule_out(non_crit_df,␣
↪f'naive_{alpha}'))
            group_aps_rule_out[alpha].append(rule_out(non_crit_df,␣
↪f'aps_{alpha}'))
            group_gaps_rule_out[alpha].append(rule_out(non_crit_df,␣
↪f'gaps_{alpha}'))
            group_raps_rule_out[alpha].append(rule_out(non_crit_df,␣
↪f'raps_{alpha}'))
            group_graps_rule_out[alpha].append(rule_out(non_crit_df,␣
↪f'graps_{alpha}'))

            group_naive_non_crit_set_size[alpha].
↪append(non_crit_df[f'naive_{alpha}'].apply(len).mean())
            group_aps_non_crit_set_size[alpha].
↪append(non_crit_df[f'aps_{alpha}'].apply(len).mean())
```

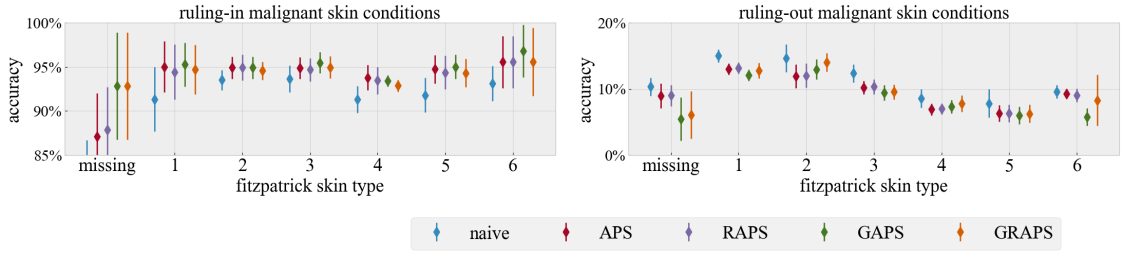
```
            group_gaps_non_crit_set_size[alpha].
↪append(non_crit_df[f'gaps_{alpha}'].apply(len).mean())
            group_raps_non_crit_set_size[alpha].
↪append(non_crit_df[f'raps_{alpha}'].apply(len).mean())
            group_graps_non_crit_set_size[alpha].
↪append(non_crit_df[f'graps_{alpha}'].apply(len).mean())

    all_group_naive_rule_in[a] = group_naive_rule_in
    all_group_aps_rule_in[a] = group_aps_rule_in
    all_group_gaps_rule_in[a] = group_gaps_rule_in
    all_group_raps_rule_in[a] = group_raps_rule_in
    all_group_graps_rule_in[a] = group_graps_rule_in

    all_group_naive_rule_out[a] = group_naive_rule_out
    all_group_aps_rule_out[a] = group_aps_rule_out
    all_group_gaps_rule_out[a] = group_gaps_rule_out
    all_group_raps_rule_out[a] = group_raps_rule_out
    all_group_graps_rule_out[a] = group_graps_rule_out

    all_group_naive_crit_set_size[a] = group_naive_crit_set_size
    all_group_aps_crit_set_size[a] = group_aps_crit_set_size
    all_group_gaps_crit_set_size[a] = group_gaps_crit_set_size
    all_group_raps_crit_set_size[a] = group_raps_crit_set_size
    all_group_graps_crit_set_size[a] = group_graps_crit_set_size

    all_group_naive_non_crit_set_size[a] = group_naive_non_crit_set_size
    all_group_aps_non_crit_set_size[a] = group_aps_non_crit_set_size
    all_group_gaps_non_crit_set_size[a] = group_gaps_non_crit_set_size
    all_group_raps_non_crit_set_size[a] = group_raps_non_crit_set_size
    all_group_graps_non_crit_set_size[a] = group_graps_non_crit_set_size
```

[63]:
```
import matplotlib.ticker as mtick
alpha = 0.10
marker = 'd'
ms = 10
# plt.ylim(0.5, 1)
fig, ax = plt.subplots(1, 2, figsize=(22, 4))
# fig.set_size_inches(14, 4)

ax[0].set_title('ruling-in malignant skin conditions', fontsize=fontsize)
ax[1].set_title('ruling-out malignant skin conditions', fontsize=fontsize)
ax[0].set_ylim(0.85, 1)
ax[1].set_ylim(0.0, 0.2)
ax[0].set_xticks(range(0, 7))
ax[1].set_xticks(range(0, 7))
ax[0].set_xticklabels(['missing', 1, 2, 3, 4, 5, 6], fontsize=fontsize)
ax[1].set_xticklabels(['missing', 1, 2, 3, 4, 5, 6], fontsize=fontsize)
```

```
ax[0].set_yticks(np.arange(0.85, 1.01, 0.05))
ax[1].set_yticks(np.arange(0, 0.201, 0.10))
ax[0].set_yticklabels(np.arange(0.85, 1.01, 0.05).round(2), fontsize=fontsize-4)
ax[1].set_yticklabels(np.arange(0, 0.201, 0.10).round(2), fontsize=fontsize-4)
ax[0].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1, decimals=0))
ax[1].yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1, decimals=0))
ax[0].set_xlabel('fitzpatrick skin type', fontsize=fontsize)
ax[1].set_xlabel('fitzpatrick skin type', fontsize=fontsize)
ax[0].set_ylabel('accuracy', fontsize=fontsize)
ax[1].set_ylabel('accuracy', fontsize=fontsize)
ax[0].errorbar(
    x=[x-0.30 for x in range(0, 7, 1)],
    y=[np.mean(list(x[alpha])) for x in all_group_naive_rule_in.values()],
    yerr=[np.std(list(x[alpha])) for x in all_group_naive_rule_in.values()],
    marker=marker,
    ms=ms,
    ls='',
    label='naive',
)
ax[0].errorbar(
    x=[x-0.15 for x in range(0, 7, 1)],
    y=[np.mean(list(x[alpha])) for x in all_group_aps_rule_in.values()],
    yerr=[np.std(list(x[alpha])) for x in all_group_aps_rule_in.values()],
    marker=marker,
    ms=ms,
    ls='',
    label='APS',
)
ax[0].errorbar(
    x=[x-0.00 for x in range(0, 7, 1)],
    y=[np.mean(list(x[alpha])) for x in all_group_raps_rule_in.values()],
    yerr=[np.std(list(x[alpha])) for x in all_group_raps_rule_in.values()],
    marker=marker,
    ms=ms,
    ls='',
    label='RAPS',
)
ax[0].errorbar(
    x=[x+0.15 for x in range(0, 7, 1)],
    y=[np.mean(list(x[alpha])) for x in all_group_gaps_rule_in.values()],
    yerr=[np.std(list(x[alpha])) for x in all_group_gaps_rule_in.values()],
    marker=marker,
    ms=ms,
    ls='',
    label='GAPS',
)
ax[0].errorbar(
```

```
    x=[x+0.30 for x in range(0, 7, 1)],
    y=[np.mean(list(x[alpha])) for x in all_group_graps_rule_in.values()],
    yerr=[np.std(list(x[alpha])) for x in all_group_graps_rule_in.values()],
    marker=marker,
    ms=ms,
    ls='',
    label='GRAPS',
)

ax[1].errorbar(
    x=[x-0.30 for x in range(0, 7, 1)],
    y=[np.mean(list(x[alpha])) for x in all_group_naive_rule_out.values()],
    yerr=[np.std(list(x[alpha])) for x in all_group_naive_rule_out.values()],
    marker=marker,
    ms=ms,
    ls='',
    label='Naive',
)
ax[1].errorbar(
    x=[x-0.15 for x in range(0, 7, 1)],
    y=[np.mean(list(x[alpha])) for x in all_group_aps_rule_out.values()],
    yerr=[np.std(list(x[alpha])) for x in all_group_aps_rule_out.values()],
    marker=marker,
    ms=ms,
    ls='',
    label='APS',
)
ax[1].errorbar(
    x=[x-0.00 for x in range(0, 7, 1)],
    y=[np.mean(list(x[alpha])) for x in all_group_raps_rule_out.values()],
    yerr=[np.std(list(x[alpha])) for x in all_group_raps_rule_out.values()],
    marker=marker,
    ms=ms,
    ls='',
    label='RAPS',
)
ax[1].errorbar(
    x=[x+0.15 for x in range(0, 7, 1)],
    y=[np.mean(list(x[alpha])) for x in all_group_gaps_rule_out.values()],
    yerr=[np.std(list(x[alpha])) for x in all_group_gaps_rule_out.values()],
    marker=marker,
    ms=ms,
    ls='',
    label='GAPS',
)
ax[1].errorbar(
    x=[x+0.30 for x in range(0, 7, 1)],
```

```
    y=[np.mean(list(x[alpha])) for x in all_group_graps_rule_out.values()],
    yerr=[np.std(list(x[alpha])) for x in all_group_graps_rule_out.values()],
    marker=marker,
    ms=ms,
    ls='',
    label='GRAPS',
)

handles, labels = ax[0].get_legend_handles_labels()
plt.tight_layout(w_pad=3, h_pad=2)
# plt.subplots_adjust(bottom=0.2)
plt.legend(
    handles,
    labels,
    # title='prediction set method',
    # title_fontproperties={
    #     'style': 'normal',
    #     'size': fontsize,
    # },
    fontsize=fontsize,
    bbox_to_anchor=(0.90, -0.40, 0.0, 0),
    ncol=7,
)
plt.savefig(fig_dir / 'fitz-rule-in-out-subgroup.pdf', bbox_inches="tight")
plt.show()
```



For our two use-cases, we find all four methods of prediction sets to perform similarly at = 0.1 in terms of accuracy and set size at a subgroup level (Figure 4). Group conformal methods (GAPS and GRAPS) in performing slightly better at ruling-out malignant skin conditions and worse at rulingout malignant skin conditions than other methods for skin types 1 and 2 (the lightest skin tones)

6.1.18 Comparing conformal uncertainty to epistemic uncertainty

[64]:
```
for _, df in test_dfs.items():
    res = []
    for i, row in df[[f'pred_{i}' for i in range(C)]].iterrows():
        res.append(torch.softmax(torch.tensor([
            getattr(row, f'pred_{i}') for i in range(C)
        ]), axis=0).numpy())
    df['mc_scores'] = res
```

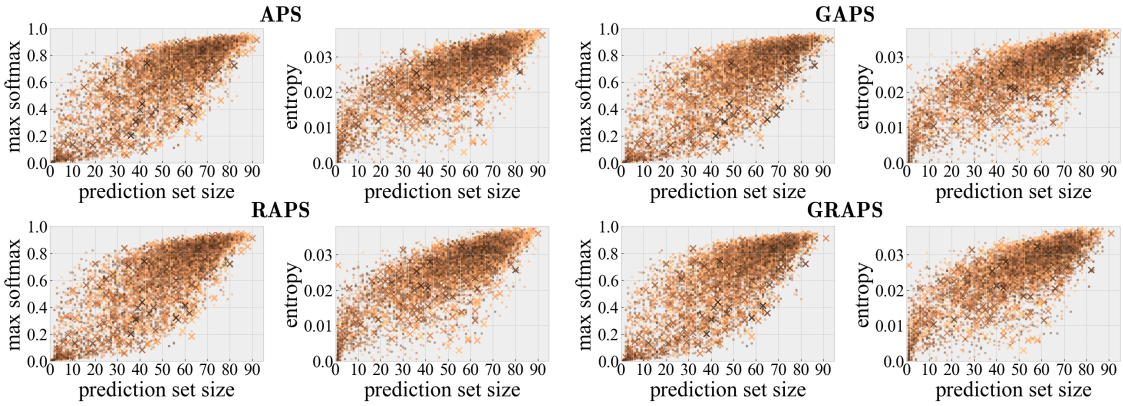
[65]:
```
for _, df in test_dfs.items():
    df['naive'] = df.mc_scores.apply(lambda x: 1 - x[:, 0].max())
    df['variance'] = df.mc_scores.apply(lambda x: np.mean(np.var(x, 1)))
    df['entropy'] = df.mc_scores.apply(lambda x: -np.mean(np.mean(x, 1) * np.
 ↪log(np.mean(x, 1))))
```

[66]:
```
subsample = 3000
fontsize=36
alpha = 0.5
pos_marker='x'
neg_marker='o'
pos_marker_size=100
neg_marker_size=12
lw=4
ls='--'
alpha_level = 0.1
threshold = 114

label_map = dict(sorted(zip(skin_df.label.unique(), itertools.count())))
malignant =[label_map[x] for x in skin_df[skin_df.three_partition_label ==␣
 ↪'malignant'].label.unique()]
pos_df = df[df.label.isin(malignant)]
neg_df = df[~df.label.isin(malignant)]
# neg_df = df[~df.label.isin(malignant)].sample(frac=0.2)

# fig, ax = plt.subplots(nrows=5, ncols=2, figsize=(14, 20), sharex=False)
fig, ax = plt.subplots(nrows=2, ncols=4, figsize=(26, 9), sharex=False)

for j, conf in enumerate([f'aps_{alpha_level}', f'raps_{alpha_level}']):
    ax[j, 0].set_ylabel('max softmax', fontsize=fontsize)
    ax[j, 1].set_ylabel('entropy', fontsize=fontsize)
    # ax[j, 2].set_ylabel('Variance', fontsize=fontsize)

    # if j == 4:
    ax[j, 0].set_xlabel('prediction set size', fontsize=fontsize)
    ax[j, 1].set_xlabel('prediction set size', fontsize=fontsize)
        # ax[j, 2].set_xlabel('Prediction set size', fontsize=fontsize)
```

```
  for i, a in enumerate(reversed(A)):
      if i == 0: continue
      c = SKIN_COLORS[i-1][None]

      sub_pos_df = pos_df.query('subgroup == @a')
      sub_pos_df = sub_pos_df.sample(n=min(subsample, len(sub_pos_df)))
      set_sizes = sub_pos_df[conf].map(len)

      sub_pos_df = sub_pos_df[set_sizes <= threshold]
      x_pos = set_sizes[set_sizes <= threshold]

      naive_y_pos = sub_pos_df.naive.values
      var_y_pos = sub_pos_df.variance.values
      ent_y_pos = sub_pos_df.entropy.values

      ax[j, 0].set_xlim(0, 95)
      ax[j, 1].set_xlim(0, 95)

      ax[j, 0].set_ylim(0, 1)
      ax[j, 1].set_ylim(0, 0.038)

      ax[j, 0].set_xticks(range(0, 100, 10), fontsize=fontsize)
      ax[j, 0].set_yticks(np.arange(0, 1.01, 0.2), fontsize=fontsize)
      ax[j, 0].set_xticklabels(range(0, 95, 10), fontsize=fontsize-8)
      ax[j, 0].set_yticklabels([round(x, 1) for x in np.arange(0, 1.01, 0.
↪2)], fontsize=fontsize-8)
      ax[j, 1].set_xticks(range(0, 100, 10), fontsize=fontsize)
      ax[j, 1].set_yticks(np.arange(0, 0.035, 0.010), fontsize=fontsize)
      ax[j, 1].set_xticklabels(range(0, 95, 10), fontsize=fontsize-8)
      ax[j, 1].set_yticklabels([round(x, 3) for x in np.arange(0, 0.035, 0.
↪010)], fontsize=fontsize-8)

      ax[j, 0].scatter(x_pos, naive_y_pos, alpha=1, c=c, s=pos_marker_size,␣
↪marker=pos_marker)
      ax[j, 1].scatter(x_pos, ent_y_pos, alpha=1, c=c, s=pos_marker_size,␣
↪marker=pos_marker)

      sub_neg_df = neg_df.query('subgroup == @a')
      sub_neg_df = sub_neg_df.sample(n=min(subsample, len(sub_neg_df)))
      set_sizes = sub_neg_df[conf].map(len)

      sub_neg_df = sub_neg_df[set_sizes <= threshold]
      x_neg = set_sizes[set_sizes <= threshold]

      naive_y_neg = sub_neg_df.naive.values
      var_y_neg = sub_neg_df.variance.values
```

```
        ent_y_neg = sub_neg_df.entropy.values

        ax[j, 0].scatter(x_neg, naive_y_neg, alpha=alpha, c=c,␣
↪s=neg_marker_size, marker=neg_marker)
        ax[j, 1].scatter(x_neg, ent_y_neg, alpha=alpha, c=c, s=neg_marker_size,␣
↪marker=neg_marker)

for j, conf in enumerate([f'gaps_{alpha_level}', f'graps_{alpha_level}']):

    ax[j, 2].set_ylabel('max softmax', fontsize=fontsize)
    ax[j, 3].set_ylabel('entropy', fontsize=fontsize)
    # ax[j, 2].set_ylabel('Variance', fontsize=fontsize)

    # if j == 4:
    ax[j, 2].set_xlabel('prediction set size', fontsize=fontsize)
    ax[j, 3].set_xlabel('prediction set size', fontsize=fontsize)
        # ax[j, 2].set_xlabel('Prediction set size', fontsize=fontsize)

    for i, a in enumerate(reversed(A)):
        if i == 0: continue
        c = SKIN_COLORS[i-1][None]

        sub_pos_df = pos_df.query('subgroup == @a')
        sub_pos_df = sub_pos_df.sample(n=min(subsample, len(sub_pos_df)))
        set_sizes = sub_pos_df[conf].map(len)

        sub_pos_df = sub_pos_df[set_sizes <= threshold]
        x_pos = set_sizes[set_sizes <= threshold]

        naive_y_pos = sub_pos_df.naive.values
        var_y_pos = sub_pos_df.variance.values
        ent_y_pos = sub_pos_df.entropy.values

        ax[j, 2].set_xlim(0, 95)
        ax[j, 3].set_xlim(0, 95)

        ax[j, 2].set_ylim(0, 1)
        ax[j, 3].set_ylim(0, 0.038)

        ax[j, 2].set_xticks(range(0, 100, 10), fontsize=fontsize)
        ax[j, 2].set_yticks(np.arange(0, 1.01, 0.2), fontsize=fontsize)
        ax[j, 2].set_xticklabels(range(0, 95, 10), fontsize=fontsize-8)
        ax[j, 2].set_yticklabels([round(x, 1) for x in np.arange(0, 1.01, 0.
↪2)], fontsize=fontsize-8)
        ax[j, 3].set_xticks(range(0, 100, 10), fontsize=fontsize)
        ax[j, 3].set_yticks(np.arange(0, 0.035, 0.010), fontsize=fontsize)
        ax[j, 3].set_xticklabels(range(0, 95, 10), fontsize=fontsize-8)
```

```
        ax[j, 3].set_yticklabels([round(x, 3) for x in np.arange(0, 0.035, 0.
↪010)], fontsize=fontsize-8)

        ax[j, 2].scatter(x_pos, naive_y_pos, alpha=1, c=c, s=pos_marker_size,␣
↪marker=pos_marker)
        ax[j, 3].scatter(x_pos, ent_y_pos, alpha=1, c=c, s=pos_marker_size,␣
↪marker=pos_marker)

        sub_neg_df = neg_df.query('subgroup == @a')
        sub_neg_df = sub_neg_df.sample(n=min(subsample, len(sub_neg_df)))
        set_sizes = sub_neg_df[conf].map(len)

        sub_neg_df = sub_neg_df[set_sizes <= threshold]
        x_neg = set_sizes[set_sizes <= threshold]

        naive_y_neg = sub_neg_df.naive.values
        var_y_neg = sub_neg_df.variance.values
        ent_y_neg = sub_neg_df.entropy.values

        ax[j, 2].scatter(x_neg, naive_y_neg, alpha=alpha, c=c,␣
↪s=neg_marker_size, marker=neg_marker)
        ax[j, 3].scatter(x_neg, ent_y_neg, alpha=alpha, c=c, s=neg_marker_size,␣
↪marker=neg_marker)

plt.figtext(0.25, 1.000, r'$\bf{APS}$', fontsize=fontsize, va='center',␣
↪ha='center')
plt.figtext(0.25, 0.50, r'$\bf{RAPS}$', fontsize=fontsize, va='center',␣
↪ha='center')
plt.figtext(0.75, 1.000, r'$\bf{GAPS}$', fontsize=fontsize, va='center',␣
↪ha='center')
plt.figtext(0.75, 0.50, r'$\bf{GRAPS}$', fontsize=fontsize, va='center',␣
↪ha='center')
plt.tight_layout(h_pad=3, w_pad=3)
plt.savefig(fig_dir / 'fitz-uncertainty-compare.png', dpi=300,␣
↪bbox_inches="tight")
plt.show()
```



Finally, we consider the relationship between set size and epistemic uncertainty, measured by maximum softmax probability and predictive entropy, and find a strong correlation between set size and epistemic uncertainty. Looking at Figure 6, we can see that the group conformal methods show increased separation between Fitzpatrick subgroups – particularly for skin type 1 and skin type 6 – compared to the naive baseline and aggregate conformal methods. This separation is quantified in Table 4, which shows that GAPS and GRAPS have lower Spearman correlation than Naive, APS, and RAPS across a range of .

[67]:
```
end_time = time.time()
print(f"Total run time: {end_time - start_time} seconds")
```

```
Total run time: 783.7835052013397 seconds
```

6.1.19 Analysis

In the content above, we saw an overall average accuracy of 0.25 +/- 0.00. The coverage decreases linearly as the alpha level increase, and the cardinality also decreases as the alpha level increase for the naive method. For all the other types based on adaptive prediction sets, although we observe the same trend, the coverage decrease less rapidly comparing to the naive method, with GAPS being the worst. For each of the skin type subgroups, we see a similar trend, which GAPS and GRAPS has slightly different cardinality values for each skin type comparing to APS and RAPS where each of the skin types shares similar set sizes, however for the coverage values, they seems to have slightly better results with the skin types shares similar coverage values.

The result above also compares the rule-in and rule-out accuracies for malignant skin conditions for example kaposi sarcoma for each of the methods and the skin types. The accuracy of ruling-out malignant skin conditions are low comparing to ruling-in, with the four proposed methods all have higher accuracy for ruling in the method. This means potentially this model is useful for early detection of the skin condition instead of diagonosing it.

6.1.20 Experiments beyond the original paper and ablation study

Not applicable to our paper of selection.

7 Discussion

Implication of Experiment results:

- Based on our experiments, we find group conformal predictors (GAPS and GRAPS) to be a promising and generally applicable approach to increasing clinical usability and trustworthiness in medical AI.

Whether the original paper was reproducible:

- Yes we think the original paper is reproducible.

"What was easy" during the reproduction

- The paper provided a Github repo containing useful links to the dataset and the notebook containing the code.
- We did not encounter much problem for example installation or dependency issues to make the notebook running.
- The dataset is accessible on the internet.

"What was difficult" during the reproduction

- Downloading the data (requires me to implement a web scraper: in order to mimic downloading from a browser, I needed to add some parameters related to browser)
- Finding a proper python version to run the code (I originally used python 3.7 and had a frustrating time setting up the environment until I switched to 3.8)
- Figuring out how to run train.py properly
- We have little pre-existing knowledge about conformal prediction and we need to understand how conformal prediction works

Recommendations to the original authors for better reproducibility:

- Record a video demo of how to get the code running.
- Include python version in the description.
- The code could be better structured with README or docstring for each of the modules.
- Add a script to download the data.
- Add a docker environment with pre-installed dependencies.
- Add a script to run the training code.

What will you do in next phase

- Figure out the claims of the result and write explanation of each of the figures produced above
- Discuss with respect to the hypothesis and the results from the original paper and make sure they make sense to us

8 Public GitHub Repo

https://github.com/lthroy/CS598DLHFinal

9 References

1. Lu, Charles, et al. “Fair conformal predictors for applications in medical imaging.” Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 36. No. 11. 2022. https://ojs.aaai.org/index.php/AAAI/article/view/21459
2. Groh, Matthew, et al. “Evaluating deep neural networks trained on clinical images in dermatology with the fitzpatrick 17k dataset.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021. https://openaccess.thecvf.com/content/CVPR2021W/ISIC/html/Groh_Evaluating_Deep_Neural_Networ