```
한국어문장분류공모전:.
├──CODE
|       DatasetMakingProcess.ipynb
|       Ensemble.ipynb
|       RoBERTa-large.ipynb
|       TPU_BERT-base.ipynb
|
└──DATASET
        result_roberta_large.csv
        result_TPU.csv
        sample_submission.csv
        test_data.csv
        train_data.csv
        train_final.csv
```

train_final 파일 만들기

```
import os
import pandas as pd
import numpy as np

import urllib.request



from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/c
```

```
train_d=pd.read_csv('/content/drive/MyDrive/데이콘영화연습대회/train_data.csv')

urllib.request.urlretrieve("https://raw.githubusercontent.com/kakaobrain/KorNLUDatasets
urllib.request.urlretrieve("https://raw.githubusercontent.com/kakaobrain/KorNLUDatasets
urllib.request.urlretrieve("https://raw.githubusercontent.com/kakaobrain/KorNLUDatasets
urllib.request.urlretrieve("https://raw.githubusercontent.com/kakaobrain/KorNLUDatasets
```

```
('xnli.test.ko.tsv', <http.client.HTTPMessage at 0x7f7d13631590>)
```

```
train_k1 = pd.read_csv("xnli.dev.ko.tsv", sep='₩t', quoting=3)
train_k2 = pd.read_csv("multinli.train.ko.tsv", sep='₩t', quoting=3)
train_k3 = pd.read_csv("snli_1.0_train.ko.tsv", sep='₩t', quoting=3)
train_k4 = pd.read_csv("xnli.test.ko.tsv", sep='₩t', quoting=3)



urllib.request.urlretrieve("https://raw.githubusercontent.com/KLUE-benchmark/KLUE/main/
urllib.request.urlretrieve("https://raw.githubusercontent.com/KLUE-benchmark/KLUE/main/
```

```
('klue-nli-v1.1_train.json', <http.client.HTTPMessage at 0x7f7d047e31d0>)
```

```
train_k5 = pd.read_json('klue-nli-v1.1_dev.json')
train_k6=pd.read_json('klue-nli-v1.1_train.json')


train_kk = pd.concat([train_k5,train_k6])
```

```
train_kk = train_kk.drop(['guid', 'genre', 'author','label2','label3','label4','label5'
```

```
train_kk.head()
```

	premise	hypothesis	gold_label
0	흡연자분들은 발코니가 있는 방이면 발코니에서 흡연이 가능합니다.	어떤 방에서도 흡연은 금지됩니다.	contradiction
1	10명이 함께 사용하기 불편함없이 만족했다.	10명이 함께 사용하기 불편함이 많았다.	contradiction
2	10명이 함께 사용하기 불편함없이 만족했다.	성인 10명이 함께 사용하기 불편함없이 없었다.	neutral
		10명이 함께 사용하기에 만족스러	..

```
train_kk.rename(columns = {'gold_label':'label'},inplace=True)
```

```
train_kk.head()
```

	premise	hypothesis	label
0	흡연자분들은 발코니가 있는 방이면 발코니에서 흡연이 가능합니다.	어떤 방에서도 흡연은 금지됩니다.	contradiction
1	10명이 함께 사용하기 불편함없이 만족했다.	10명이 함께 사용하기 불편함이 많았다.	contradiction
2	10명이 함께 사용하기 불편함없이 만족했다.	성인 10명이 함께 사용하기 불편함없이 없었다.	neutral
		10명이 함께 사용하기에 만족스러	..

```
train_k = pd.concat([train_k1,train_k2,train_k3,train_k4],axis=0)
```

```
train_k.head()
```

	sentence1	sentence2	gold_label
0	그리고 그가 말했다, "엄마, 저 왔어요."	그는 학교 버스가 그를 내려주자마자 엄마에게 전화를 걸었다.	neutral
1	그리고 그가 말했다, "엄마, 저 왔어요."	그는 한마디도 하지 않았다.	contradiction
2	그리고 그가 말했다, "엄마, 저 왔어요."	그는 엄마에게 집에 갔다고 말했다.	entailment
3	내가 무엇을 위해 가고 있는지 또는 어떤 것을 위해 있는지 몰랐기 때문에 워싱턴의 ...	나는 워싱턴에 가본 적이 없어서 거기 배정을 받았을 때 그 장소를 찾으려다가 길을 ...	neutral

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-18-f7038a60e200> in <module>()
----> 1 train_k.drop(['premise','hypothesis','label'])

                                3 frames
/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py in _drop_axis(self, labels, axis, level, errors)
   4212                 labels_missing = (axis.get_indexer_for(labels) == -1).any()
   4213                 if errors == "raise" and labels_missing:
-> 4214                     raise KeyError(f"{labels} not found in axis")
   4215
   4216             slicer = [slice(None)] * self.ndim

KeyError: "['premise' 'hypothesis' 'label'] not found in axis"
```

```
train_k.rename(columns = {'sentence1':'premise', 'sentence2':'hypothesis' ,'gold_label'
```

```
train_k.head()
```

	premise	hypothesis	label
0	그리고 그가 말했다, "엄마, 저 왔어요."	그는 학교 버스가 그를 내려주자마자 엄마에게 전화를 걸었다.	neutral
1	그리고 그가 말했다, "엄마, 저 왔어요."	그는 한마디도 하지 않았다.	contradiction
2	그리고 그가 말했다, "엄마, 저 왔어요."	그는 엄마에게 집에 갔다고 말했다.	entailment
3	내가 무엇을 위해 가고 있는지 또는 어떤 것을 위해 있는지 몰랐기 때문에 워싱턴의 …	나는 워싱턴에 가본 적이 없어서 거기 배정을 받았을 때 그 장소를 찾으려다가 길을 …	neutral

```
train=pd.concat([train_d,train_k,train_kk], axis=0)
```

```
train = train.sample(frac=1)
```

```
train.head()
```

	index	premise	hypothesis	label
192369	NaN	검은 개가 시냇물에서 천을 꺼낸다.	개 한 마리가 개울로 들어간다.	entailment
375961	NaN	감자로 식사를 준비하는 두 사람, 한 남자와 한 여자의 머리 위 풍경.	두 사람이 가족을 위해 식사를 준비하고 있다.	neutral
542770	NaN	한 남자와 그의 가족은 심야 간식을 먹고 있다.	어떤 사람들은 간식을 먹고 있다.	entailment
		베이스 기타를 연주하고 공중으로 발	수컷이 팔굽혀펴기를 하고	

```
train.drop(['index'], inplace=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-30-ace8a3218b09> in <module>()
----> 1 train.drop(['index'], inplace=True)

                                3 frames
/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py in _drop_axis(self, labels,
axis, level, errors)
   4212                 labels_missing = (axis.get_indexer_for(labels) == -1).any()
   4213                 if errors == "raise" and labels_missing:
-> 4214                     raise KeyError(f"{labels} not found in axis")
   4215
   4216             slicer = [slice(None)] * self.ndim

KeyError: "['index'] not found in axis"
```

```
train.to_csv("train_final.csv", index=False,encoding='utf-8')
```

0초 오후 4:14에 완료됨

RoBERTa-large

Import Pakages

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')
```

Load Data

```
train = pd.read_csv("drive/MyDrive/한국어문장분류공모전/DATA/train_data.csv")
test = pd.read_csv("drive/MyDrive/한국어문장분류공모전/DATA/test_data.csv")
submission = pd.read_csv("drive/MyDrive/한국어문장분류공모전/DATA/sample_submission.csv")

train.head(5)
```

Train, Test Data 확인

```
print(train.info(), end='₩n₩n')
print(test.info())
```

```
print('Train Columns: ', train.columns)
print('Test Columns: ', test.columns)
```

```
print('Train Label: ', train['label'].value_counts(), sep='₩n', end='₩n₩n')
print('Test Label: ', test['label'].value_counts(), sep='₩n')
```

```
print('Train Null: ', train.isnull().sum(), sep='₩n', end='₩n₩n')
print('Test Null: ', test.isnull().sum(), sep='₩n')
```

Preprocessing

```
train['premise'] = train['premise'].str.replace('[^ㄱ-ㅎㅏ-ㅣ가-힣 0-9]', '')
test['premise'] = test['premise'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 0-9]", "")
```

```
train.head(5)
```

```
train['hypothesis'] = train['hypothesis'].str.replace('[^ㄱ-ㅎㅏ-ㅣ가-힣 0-9]', '')
test['hypothesis'] = test['hypothesis'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 0-9]", "")
train.head(5)
```

Modeling

```
!pip install transformers
```

```
import random
from tqdm import tqdm

import torch
import torch.nn.functional as F
from torch.utils.data import DataLoader
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from transformers import TrainingArguments, Trainer
from transformers import AutoModelForSequenceClassification, AutoConfig, AutoTokenizer
```

Load Tokenizer and Model

```
MODEL_NAME = 'klue/roberta-large'

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

config = AutoConfig.from_pretrained(MODEL_NAME)
config.num_labels = 3

model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME, config=config)

print(model)
print(config)
```

Tokenizing

```
train_dataset, eval_dataset = train_test_split(train, test_size=0.2, shuffle=True, stratify=train['

tokenized_train = tokenizer(
    list(train_dataset['premise']),
    list(train_dataset['hypothesis']),
    return_tensors="pt"
```

```
    return_tensors= pt",
    max_length=256, # Max_Length = 190
    padding=True,
    truncation=True,
    add_special_tokens=True
)

tokenized_eval = tokenizer(
    list(eval_dataset['premise']),
    list(eval_dataset['hypothesis']),
    return_tensors="pt",
    max_length=256,
    padding=True,
    truncation=True,
    add_special_tokens=True
)

print(tokenized_train['input_ids'][0])
print(tokenizer.decode(tokenized_train['input_ids'][0]))
```

```
class BERTDataset(torch.utils.data.Dataset):
    def __init__(self, pair_dataset, label):
        self.pair_dataset = pair_dataset
        self.label = label

    def __getitem__(self, idx):
        item = {key: val[idx].clone().detach() for key, val in self.pair_dataset.items()}
        item['label'] = torch.tensor(self.label[idx])

        return item

    def __len__(self):
        return len(self.label)
```

```
def label_to_num(label):
    label_dict = {"entailment": 0, "contradiction": 1, "neutral": 2, "answer": 3}
    num_label = []

    for v in label:
        num_label.append(label_dict[v])

    return num_label


train_label = label_to_num(train_dataset['label'].values)
eval_label = label_to_num(eval_dataset['label'].values)
```

```
train_dataset = BERTDataset(tokenized_train, train_label)
eval_dataset = BERTDataset(tokenized_eval, eval_label)

print(train_dataset.__len__())
print(train_dataset.__getitem__(19997))
print(tokenizer.decode(train_dataset.__getitem__(19997)['input_ids']))
```

```
print(tokenizer.decode(train_dataset.__getitem__(19997)['input_ids']))
```

```
def compute_metrics(pred):
  """ validation을 위한 metrics function """
  labels = pred.label_ids
  preds = pred.predictions.argmax(-1)
  probs = pred.predictions

  # calculate accuracy using sklearn's function
  acc = accuracy_score(labels, preds) # 리더보드 평가에는 포함되지 않습니다.

  return {
      'accuracy': acc,
  }
```

Fine Tuning

```
training_ars = TrainingArguments(
    output_dir='./result',
    num_train_epochs=7,
    per_device_train_batch_size=128,
    save_total_limit=5,
    save_steps=500,
    evaluation_strategy='steps',
    eval_steps = 500,
    load_best_model_at_end = True,
)

trainer = Trainer(
    model=model,
    args=training_ars,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)
```

```
trainer.train()
model.save_pretrained('./result/best_model')
```

결과 추론

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

Tokenizer_NAME = "klue/roberta-large"
tokenizer = AutoTokenizer.from_pretrained(Tokenizer_NAME)
```

```
MODEL_NAME = './result/checkpoint-4000'
model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME)
model.resize_token_embeddings(tokenizer.vocab_size)
model.to(device)

print(tokenizer)
```

```
test_label = label_to_num(test['label'].values)

tokenized_test = tokenizer(
    list(test['premise']),
    list(test['hypothesis']),
    return_tensors="pt",
    max_length=128,
    padding=True,
    truncation=True,
    add_special_tokens=True
)

test_dataset = BERTDataset(tokenized_test, test_label)

print(test_dataset.__len__())
print(test_dataset.__getitem__(1665))
print(tokenizer.decode(test_dataset.__getitem__(6)['input_ids']))
```

```
dataloader = DataLoader(test_dataset, batch_size=16, shuffle=False)

model.eval()
output_pred = []
output_prob = []

for i, data in enumerate(tqdm(dataloader)):
    with torch.no_grad():
        outputs = model(
            input_ids=data['input_ids'].to(device),
            attention_mask=data['attention_mask'].to(device),
            token_type_ids=data['token_type_ids'].to(device)
        )
    logits = outputs[0]
    prob = F.softmax(logits, dim=-1).detach().cpu().numpy()
    logits = logits.detach().cpu().numpy()
    result = np.argmax(logits, axis=-1)

    output_pred.append(result)
    output_prob.append(prob)


pred_answer, output_prob = np.concatenate(output_pred).tolist(), np.concatenate(output_prob, axis=0
print(pred_answer)
```

```
def num_to_label(label):
    label_dict = {0: "entailment", 1: "contradiction", 2: "neutral"}
    str_label = []
```

```
    for i, v in enumerate(label):
        str_label.append([i,label_dict[v]])

    return str_label

answer = num_to_label(pred_answer)
print(answer)
```

```
df = pd.DataFrame(answer, columns=['index', 'label'])

df.to_csv('./result/result_roberta_large.csv', index=False)

print(df)
```

BERT-base

Install transformer

```
pip install transformers
```

Import libraries

```
import os
import pandas as pd
import numpy as np
from tqdm import tqdm
import urllib.request
from sklearn import preprocessing
import tensorflow as tf
from transformers import BertTokenizer, TFBertModel
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

Dataset Load

```
from google.colab import drive
drive.mount('/content/drive')
```

```
train=pd.read_csv('/content/drive/MyDrive/한국어문장분류공모전/DATA/train_final.csv')
test_data = pd.read_csv('/content/drive/MyDrive/한국어문장분류공모전/DATA/test_data.csv')
submission_data = pd.read_csv("/content/drive/MyDrive/한국어문장분류공모전/DATA/sample_submission.c
```

```
train_data=train[:797848]
val_data=train[797848:]
```

```
len(train_data)
```

```
train_data.head()
```

Dataset 개수 확인

```
train_data.head()
```

```
train_data.head()
```

```
print('훈련용 샘플 개수 :',len(train_data))
print('검증용 샘플 개수 :',len(val_data))
print('테스트용 샘플 개수 :',len(test_data))
```

Tokenizing

```
tokenizer = BertTokenizer.from_pretrained("klue/bert-base")
```

```
max_seq_len = 128
```

```
prem = train_data['premise'].iloc[0]
hypo = train_data['hypothesis'].iloc[0]

print(prem)
print(hypo)
```

```
encoding_result = tokenizer.encode_plus(prem, hypo, max_length=max_seq_len, pad_to_max_length=True)
```

```
print(encoding_result['input_ids'])
```

```
print(encoding_result['token_type_ids'])
```

```
def convert_examples_to_features(prem_list, hypo_list, max_seq_len, tokenizer):

    input_ids, attention_masks, token_type_ids = [], [], []

    for prem, hypo in tqdm(zip(prem_list, hypo_list), total=len(prem_list)):
        encoding_result = tokenizer.encode_plus(prem, hypo, max_length=max_seq_len, pad_to_max_leng

        input_ids.append(encoding_result['input_ids'])
        attention_masks.append(encoding_result['attention_mask'])
        token_type_ids.append(encoding_result['token_type_ids'])

    input_ids = np.array(input_ids, dtype=int)
    attention_masks = np.array(attention_masks, dtype=int)

    token_type_ids = np.array(token_type_ids, dtype=int)

    return (input_ids, attention_masks, token_type_ids)
```

```
X_train = convert_examples_to_features(train_data['premise'], train_data['hypothesis'], max_seq_len
```

```
# 최대 길이: 128
input_id = X_train[0][0]
attention_mask = X_train[1][0]
token_type_id = X_train[2][0]

print('단어에 대한 정수 인코딩 :',input_id)
print('어텐션 마스크 :',attention_mask)
print('세그먼트 인코딩 :',token_type_id)
print('각 인코딩의 길이 :', len(input_id))
print('정수 인코딩 복원 :',tokenizer.decode(input_id))
```

```
X_val = convert_examples_to_features(val_data['premise'], val_data['hypothesis'], max_seq_len=max_s
```

```
input_id = X_val[0][0]
attention_mask = X_val[1][0]
token_type_id = X_val[2][0]

print('단어에 대한 정수 인코딩 :',input_id)
print('어텐션 마스크 :',attention_mask)
print('세그먼트 인코딩 :',token_type_id)
print('각 인코딩의 길이 :', len(input_id))
print('정수 인코딩 복원 :',tokenizer.decode(input_id))
```

```
X_test = convert_examples_to_features(test_data['premise'], test_data['hypothesis'], max_seq_len=ma
```

```
train_label = train_data['label'].tolist()
val_label = val_data['label'].tolist()
test_label = test_data['label'].tolist()
```

```
idx_encode = preprocessing.LabelEncoder()
idx_encode.fit(train_label)

y_train = idx_encode.transform(train_label) # 주어진 고유한 정수로 변환
y_val = idx_encode.transform(val_label) # 고유한 정수로 변환
# y_test = idx_encode.transform(test_label) # 고유한 정수로 변환

label_idx = dict(zip(list(idx_encode.classes_), idx_encode.transform(list(idx_encode.classes_))))
idx_label = {value: key for key, value in label_idx.items()}
print(label_idx)
print(idx_label)
```

Fine Tuning

```
class TFBertForSequenceClassification(tf.keras.Model):
    def __init__(self, model_name, num_labels):
        super(TFBertForSequenceClassification, self).__init__()
        self.bert = TFBertModel.from_pretrained(model_name, from_pt=True)
        self.classifier = tf.keras.layers.Dense(num_labels,
```

```
                                                kernel_initializer=tf.keras.initializers.TruncatedN
                                                activation='softmax',
                                                name='classifier')

    def call(self, inputs):
        input_ids, attention_mask, token_type_ids = inputs
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=toke
        cls_token = outputs[1]
        prediction = self.classifier(cls_token)

        return prediction
```

TPU 작동을 위한 코드

```
resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu='grpc://' + os.environ['COLAB_TPU_
tf.config.experimental_connect_to_cluster(resolver)
tf.tpu.experimental.initialize_tpu_system(resolver)
```

```
strategy = tf.distribute.experimental.TPUStrategy(resolver)
```

```
with strategy.scope():
  model = TFBertForSequenceClassification("klue/bert-base", num_labels=3)
  optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
  loss = tf.keras.losses.SparseCategoricalCrossentropy()
  model.compile(optimizer=optimizer, loss=loss, metrics = ['accuracy'])
```

```
early_stopping = EarlyStopping(
    monitor="val_accuracy",
    min_delta=0.001,
    patience=2)

model.fit(
    X_train, y_train, epochs=5, batch_size=512, validation_data = (X_val, y_val),
    callbacks = [early_stopping]
)
```

```
model.summary()
```

결과 추론

```
pred=model.predict(X_test)
len(pred)
```

```
result=[np.argmax(val) for val in pred]
len(result)
result[0:10]
```

```
out = [list(label_idx.keys())[_] for _ in result]
out[:3]
```

```
submission_data['label']=out
submission_data.to_csv("result_TPU.csv", index=False)
```

Model Ensemble

PORORO pretrained model

```
# pororo only supports python>=3.6
!python -V
```

```
!wget https://www.python.org/ftp/python/3.6.0/Python-3.6.0.tgz
!tar xvfz Python-3.6.0.tgz
!Python-3.6.0/configure
!make
!sudo make install
```

```
!pip install pororo
```

```
!git clone https://github.com/kakaobrain/pororo.git
!cd pororo
!pip install -e .
```

```
from google.colab import drive
import pandas as pd
drive.mount('/content/drive')

test_data = pd.read_csv('/content/drive/MyDrive/한국어문장분류공모전/DATA/test_data.csv')
submission_data = pd.read_csv("/content/drive/MyDrive/한국어문장분류공모전/DATA/sample_submission.c
```

```
premise, hypothesis = list(test_data['premise']), list(test_data['hypothesis'])
```

```
from pororo import Pororo
Pororo.available_tasks()
nli_module = Pororo(task="nli", lang="ko")
nli_module(premise[0], hypothesis[0])
```

```
!pip install tqdm
```

```
from tqdm import tqdm_notebook


nli_results = []

for pre, hypo in tqdm_notebook(zip(premise, hypothesis)):
  result = nli_module(pre, hypo)

  nli_results.append(result)
```

```
submission_data['label'] = nli_results
submission_data.head(5)
```

```
import numpy as np

def label_to_num(label):
    label_dict = {"Entailment":0, "entailment":0, "Contradiction":1, "contradiction":1, "Neutral":2
    str_label = []

    for i, v in enumerate(label):
        str_label.append([i,label_dict[v]])

    return str_label
```

```
answer = np.array(label_to_num(submission_data['label']))
answer_pororo = np.array(range(1666))
for i in range(0,1666):
  answer_pororo[i] = answer[i][1]
print(answer_pororo[0:10])
```

Huffon/klue-roberta-base-nli pretrained model

```
from transformers import pipeline, AutoTokenizer
```

```
tokenizer = AutoTokenizer.from_pretrained("Huffon/klue-roberta-base-nli")
model = pipeline("text-classification", model="Huffon/klue-roberta-base-nli", return_all_scores=Fal
```

```
test_data = pd.read_csv('/content/drive/MyDrive/한국어문장분류공모전/DATA/test_data.csv')
X_test = test_data["premise"] + " " + tokenizer.sep_token + " " + test_data["hypothesis"]
```

```
from tqdm import tqdm
y_preds = []
for i in tqdm(range(X_test.shape[0])):
    y_pred = model(X_test[i])[0]["label"].lower()
    y_preds.append(y_pred)

y_preds = np.array(y_preds)
```

```
submission_data["label"] = y_preds
submission = submission_data.loc[:, ["index", "label"]]
```

```
answer = np.array(label_to_num(submission['label']))
answer_Huffon = np.array(range(1666))
```

```
for i in range(0,1666):
  answer_Huffon[i] = answer[i][1]
print(answer_Huffon[0:10])
```

Fine-tuned Huggingface roberta-large 모델 결과파일 로드

```
df_roberta_large = pd.read_csv('/content/drive/MyDrive/한국어문장분류공모전/DATA/result_roberta_lar
answer = np.array(label_to_num(df_roberta_large['label']))
answer_roberta_large = np.array(range(1666))
for i in range(0,1666):
  answer_roberta_large[i] = answer[i][1]
print(answer_roberta_large[0:10])
```

Fine-tuned Huggingface bert-base 모델 결과파일 로드

```
df_TPU = pd.read_csv('/content/drive/MyDrive/한국어문장분류공모전/DATA/result_TPU.csv')
answer = np.array(label_to_num(df_TPU['label']))
answer_TPU = np.array(range(1666))
for i in range(0,1666):
  answer_TPU[i] = answer[i][1]
print(answer_TPU[0:10])
```

Ensemble

```
print(answer_pororo.shape)
print(answer_Huffon.shape)
print(answer_roberta_large.shape)
print(answer_TPU.shape)
answer_final = np.array([answer_pororo, answer_Huffon, answer_roberta_large, answer_TPU])
print(answer_final.shape)
```

```
submission_np = np.array(range(1666))
for i in range(1666):
  vals,counts = np.unique([answer_final[0][i], answer_final[1][i], answer_final[2][i], answer_final
  index = np.argmax(counts)
  submission_np[i] = vals[index]


submission_np[0:10]
```

```
def num_to_label(num):
    label_dict = {0:"entailment", 1:"contradiction", 2:"neutral"}
    str_label = []
```

```
    for i, v in enumerate(num):
        str_label.append([i,label_dict[v]])

    return str_label
```

```
submission_sample = num_to_label(submission_np)
```

```
submission_sample[0:10]
```

```
df = pd.DataFrame(submission_sample, columns=['index', 'label'])

df.to_csv('./Cornsomazing0228.csv', index=False)

print(df)
```