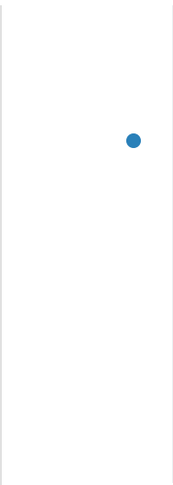

FrankWolfeFrameWork.ipynb

FrankWolfeFrameWork.ipynb

```python
import pandas as pd
import numpy as np
import torch
import torch.optim as optim
import torch.nn.functional as F
import matplotlib.pyplot as plt
import torch.nn.init as init
import torch.optim.lr_scheduler as lr_scheduler
from sklearn.preprocessing import MinMaxScaler
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
```

- pandas numpy
- torch
- matplotlib.pyplot
- sklearn.preprocessing.MinMaxScaler

```python
if torch.cuda.is_available():
    device = torch.device("cuda")
```

```
    print("GPU available:", device)
    torch.cuda.init()
else:
    device = torch.device("cpu")
    print("GPU unavailable: CPU")
```

- `torch.device`

`get_offer_data`

python

```
def get_offer_data(data_para):
    offerset_list = []
    sell_list = []
    mask_list = []
    max_num = 32
    for srch_id, group in data_para:
        num_product = len(group)
        # parallel offerset size
        offerset = group.drop(columns=['booking_bool', 'srch_id']).values
        offer_dummy = np.zeros((max_num - num_product, offerset.shape[1]))
        offerset = np.vstack((offerset, offer_dummy))
        offer_valid_mask = np.append(np.ones(num_product), np.zeros(max_n

        # parallel offerset market share
        num_sell = group['booking_bool'].values
        num_sell_dummy = np.zeros((max_num - num_product))
        num_sell = np.hstack((num_sell, num_sell_dummy))

        offerset_list.append(offerset)
        sell_list.append(num_sell)
        mask_list.append(offer_valid_mask)

    offerset_list = np.array(offerset_list)
    mask_list = np.array(mask_list)
    sell_list = np.array(sell_list)

    return offerset_list, sell_list, mask_list
```

- `data_para` `srch_id`
- `max_num = 32` `srch_id`
- `offerset_list` `sell_list` `booking_bool` `mask_list`

---

python

```
class CustomDataset(Dataset):
    def __init__(self, offerset_tensor, sell_tensor, mask_tensor):
        self.offerset_tensor = offerset_tensor
```

```
        self.sell_tensor = sell_tensor
        self.mask_tensor = mask_tensor

    def __len__(self):
        return len(self.offerset_tensor)

    def __getitem__(self, idx):
        return self.offerset_tensor[idx], self.sell_tensor[idx], self.mask_tensor
```

- `torch.utils.data.Dataset`
- `offerset_tensor` `sell_tensor` `mask_tensor`

`offer set`

`Sales`

```python
search_info = ['srch_id']
continuous_feature = ['position', 'prop_starrating', 'prop_location_score1',
                      'prop_log_historical_price', 'srch_booking_window',
                      'srch_length_of_stay', 'srch_adults_count',
                      'srch_children_count', 'srch_room_count', 'price_usd']

discrete_feature = ['prop_brand_bool', 'promotion_flag',
          'srch_saturday_night_bool', 'random_bool', 'booking_bool']
```

- `search_info` `srch_id`
- `continuous_feature` `position` `prop_starrating` `price_usd`
- `discrete_feature` `promotion_flag` `random_bool`

```python
```

```
tr_data = pd.read_csv('/content/drive/MyDrive/Choice Model/train_28-32_1
te_data = pd.read_csv('/content/drive/MyDrive/Choice Model/test_28-32_10
```

- `train_28-32_10000.csv` `test_28-32_1000.csv`

```python
scaler = MinMaxScaler()
scaler.fit(tr_data[continuous_feature])
tr_data[continuous_feature] = scaler.transform(tr_data[continuous_feature
te_data[continuous_feature] = scaler.transform(te_data[continuous_featur
```

- `MinMaxScaler()` `[0,1]`

```python
tr_data = tr_data[search_info + continuous_feature + discrete_feature]
te_data = te_data[search_info + continuous_feature + discrete_feature]
```

- `search_info` `continuous_feature` `discrete_feature`

`offer set`

```python
tr_offerset, tr_sell, tr_mask = get_offer_data(tr_data.groupby('srch_id'))
te_offerset_list, te_sell_list, te_mask_list = get_offer_data(te_data.groupby('s
```

- `get_offer_data`
  - `srch_id` `srch_id`
  - `offerset` `sell` `mask`
  -

```python
tr_offerset.shape, tr_sell.shape, tr_mask.shape
```

- offerset sell mask

Sales

```python
class Sales:
    def __init__(self, p_offerset, p_sell, p_mask):
        # N_sales (num_offers, num_products)
        self.N_sales = torch.tensor(p_sell, dtype=torch.float32)
        # offerset (num_offers, num_products, num_features)
        self.offerset = torch.tensor(p_offerset, dtype=torch.float32)
        # mask (num_offers, num_products)
        self.mask = torch.tensor(p_mask, dtype=torch.float32)

        # Some key information
        self.num_offers = self.offerset.shape[0]
        self.num_products = self.offerset.shape[1]
        self.num_features = self.offerset.shape[2]
        self.N = torch.sum(self.N_sales)
```

- N_sales (num_offers, num_products)

- offerset (num_offers, num_products, num_features)
- mask
- num_offers, num_products, num_features
- N = torch.sum(self.N_sales)

---

offer set

Sales

Preference

```python
class Preference(nn.Module):
    def __init__(self, p_num_feature):
        super(Preference, self).__init__()
        self.linear = nn.Linear(p_num_feature, 1)

    def forward(self, p_offerset, p_mask):
        output = self.linear(p_offerset).squeeze(-1)
        masked_e = torch.where(p_mask == 1, output, float('-inf'))
        log_choice_p = F.log_softmax(masked_e, dim=-1)
        return log_choice_p
```

- nn.Module
- self.linear = nn.Linear(p_num_feature, 1)
  - p_num_feature
- masked_e = torch.where(p_mask == 1, output, float('-inf'))
  - -inf softmax
- log_choice_p = F.log_softmax(masked_e, dim=-1)
  -

Problem_FrankWolfe

Problem_FrankWolfe

-
-
-

__init__

```python
class Problem_FrankWolfe:
    def __init__(self, p_offerset, p_sell, p_mask):
```

```
        self.sales = Sales(p_offerset, p_sell, p_mask)
        self.dataset = CustomDataset(self.sales.offerset, self.sales.N_sales, se
        self.train_loader = DataLoader(self.dataset, batch_size=1024, shuffle=

        self.NLL_main = None  #
        self.g = None  #
        self.NLL_gradient = None  #
        self.fw_list = []  #
        self.taste_list = []  #
        self.proportion = [1]  #
```

initialize

python

```
def initialize(self):
    initial_preference = Preference(self.sales.num_features)
    print('Initial Training Begin')
    criterion = nn.NLLLoss()
    optimizer = optim.Adam(initial_preference.parameters(), lr=5e-3)

    for epoch in range(300):
        for batch_idx, (offerset, sell, mask) in enumerate(self.train_loader):
            log_choice_p = initial_preference(offerset, mask)
            sell = sell.type(torch.int64).argmax(dim=1)
            loss = criterion(log_choice_p, sell)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

    print('Initial Training End')

    #
    self.taste_list.append(initial_preference)
    self.fw_list.append(torch.exp(log_choice_p))

    #
    self.main_problem_loss()
```

- Preference
- self.taste_list.append(initial_preference)
- self.fw_list.append(torch.exp(log_choice_p))

---

main_problem_loss

python

```
def main_problem_loss(self):
    N_sales = self.sales.N_sales
    with torch.no_grad():
        f = torch.zeros(N_sales.shape, dtype=torch.float32, device=device)
        for proportion, fw in zip(self.proportion, self.fw_list):
            f += proportion * fw
    f.requires_grad = True
    f_log = torch.log(f)
```

```
    self.NLL_main = nn.NLLLoss()(f_log, N_sales.type(torch.int64).argmax(dir
    self.NLL_main.backward()
    self.NLL_gradient = f.grad.clone()
```

- `f = sum(proportion * fw)`
- `NLL_main`
- `self.NLL_gradient`

`support_finding`

python

```
def support_finding(self):
    print('-----Consumer Type Search Begin-----')
    new_preference = Preference(self.sales.num_features)
    criterion = self.support_finding_loss
    optimizer = optim.Adam(new_preference.parameters(), lr=5e-2)

    for epoch in range(500):
        for batch_idx, (offerset, gradient, mask) in enumerate(self.train_loade
            log_choice_p = new_preference(offerset, mask)
            choice_p = torch.exp(log_choice_p)
            loss = criterion(choice_p, gradient)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

    self.taste_list.append(new_preference)
    self.fw_list.append(torch.exp(log_choice_p))
    print('-----Consumer Type Search End-----')
```

- `Preference` `NLL_gradient`
-

`proportion_update`

python

```
def proportion_update(self):
    print('-----Proportion Update Search Begin-----')
    alpha = torch.empty((len(self.taste_list), 1, 1), dtype=torch.float32, requi
    init.uniform_(alpha, 0, 0)
    optimizer = optim.Adam([alpha], lr=5e-3)

    for epoch in range(3000):
        loss = self.proportion_update_loss(alpha)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```
    self.proportion = F.softmax(alpha, dim=0).flatten().tolist()
    print('-----Proportion Update Search End-----')
```

- 1
-

Preference NLL

- support_finding Preference
- proportion_update α
- main_problem_loss NLL