MSDS 630 Advanced Machine Learning University of San Francisco

---

In case of any errors on displaying the math formulas, please use this this file for reviewing instead.

README.pdf

1 Recommender System

1.0 Video and Paper Problems

1.0.1 Video Problems

Ref: https://www.youtube.com/watch?v=zzTbptEdKhY

- What the video is about?

Answer: How to recommend content to users in YouTube.

- Decribe a little bit about the initial neural network.

Answer: Inputs are embedded videos watched and embedded search tokens with geographic information. There are three ReLU layers with a KNN and a softmax in the end.

- What is the cold start problem in this video?

Answer: If you have a new video, there will not be much signals or information to train the neural networks.

- What is one solution of the cold start problem?

Answer: The topic of the video would help.

- How they modify the initial model to solve the cold start problem?

Answer: Adding topic vector to the input.

- What's the method they use to generate topic vector?

Answer: Deep CNN, which means to train on the video shots or audio.

- How to maintain a huge list of topics?

Answer: Use a structured knowledge graph.

- What are the components of triples?

Answer: Entity, property, value.

- What is the solution if the names are not exactly match?

Answer: Look at all the hyperlinks inside wikipedia and these always links to many different names.

- How to decide which topic is the central topic of a video?

Answer: Looks at co-occurrence of topics on wikipedia, so some topics get more votes.

- Describe the steps of entity linking using textual metadata.

Answer: mention detection, disambiguation, pruning.

1.0.2 Paper Problems

Ref: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45530.pdf

- What are the steps for building this recommendation system?

```
candidate generate, scoring, reranking
```

- What are the two models they use in building this recommendation system?

```
candidate generation model, ranking model
```

- Why do we need a ranking model?

```
By using a ranking model, they reduced the number of the recommendations based on candidate generations from hundreds to dozens, which makes it easier for users to browse.
```

1.1 Modeling Theory Problems

- What is gradient descent?

Answer: gradient descent is a iterative optimization algorithm for finding the minimum of a function.

- Suppose we are considering a matrix factorization model with `n` users and `m` items, with a embedding size of `K`. Then how many parameters show we have in this model.

```
K * (m + n)
```

- Given the gradient descent equations of matrix factorization with MSE in algebra expressions.

$$u_{ik} \leftarrow u_{ik} + \frac{2\eta}{N} \sum_{j:r_{ij}=1} (y_{ij} - u_i \cdot v_j) v_{jk}$$
$$v_{jk} \leftarrow v_{jk} + \frac{2\eta}{N} \sum_{i:r_{ij}=1} (y_{ij} - u_i \cdot v_j) u_{ik}$$

- Given the gradient descent equations of matrix factorization with bias using MSE as the loss in algebra expressions.

$$u_{ik} \leftarrow u_{ik} + \frac{2\eta}{N} \sum_{j:r_{ij}=1} (y_{ij} - u_i \cdot v_j - u_{0i} - v_{0j}) v_{jk}$$
$$v_{jk} \leftarrow v_{jk} + \frac{2\eta}{N} \sum_{i:r_{ij}=1} (y_{ij} - u_i \cdot v_j - u_{0i} - v_{0j}) u_{ik}$$

$$u_{0i} \leftarrow u_{0i} + \frac{2\eta}{N} \sum_{j:r_{ij}=1} (y_{ij} - u_i \cdot v_j - u_{0i} - v_{0j})$$
$$v_{0j} \leftarrow v_{0j} + \frac{2\eta}{N} \sum_{i:r_{ij}=1} (y_{ij} - u_i \cdot v_j - u_{0i} - v_{0j})$$

- Given the gradient descent equations of matrix factorization with L2 regularization using MSE as the loss in algebra expressions.

$$u_{ik} \leftarrow u_{ik} + \frac{2\eta}{N} \left(\sum_{j:r_{ij}=1} (y_{ij} - u_i \cdot v_j) v_{jk} + \lambda \sum_{i=1}^{n_u} \sum_{k=1}^{K} u_{ik} \right)$$
$$v_{jk} \leftarrow v_{jk} + \frac{2\eta}{N} \left(\sum_{i:r_{ij}=1} (y_{ij} - u_i \cdot v_j) u_{ik} + \lambda \sum_{i=1}^{n_m} \sum_{k=1}^{K} v_{jk} \right)$$

- Given the gradient equations of matrix factorization in matrix expressions.

$$\frac{\partial E}{\partial U} = -\frac{2}{N} \Delta \cdot V$$
$$\frac{\partial E}{\partial V} = -\frac{2}{N} \Delta^T \cdot U$$

Where,

$$\Delta = (Y - U \cdot V^T) \otimes R$$

- Given a loss function used for optimizing matrix factorization.

$$E(U, V) = \frac{1}{N} \sum_{(i,j):r_{ij}=1} (y_{ij} - u_i \cdot v_j)^2$$

- How would you use a content based recommendation system for online blogs?

Answer: Analyze content of the blogs, then use the similarity between blogs to recommend blogs similar to what a user likes.

- How would you use a collaborative filtering recommendation system for online blogs?

Answer: Use past user behaviours and similarities between users and items simultaneously to provide recommendations.

- Select "explicit feedback" (E) or "implicit feedback" (I) for the following user data.

```
rating
clicks
transactions
purchases
navgigation history
```

Solution:

```
rating                  E
clicks                  I
transactions            I
purchases               I
navgigation history     I
```

- Explain the difference between implicit feedback or explicit feedback?

```
- Implicit rating: make inference from user's behavior
- Explicit rating: ask users to rate items
```

- What are the benefits and drawbacks for implicit feedbacks?

```
- Benefits:
  - available and easy to get

- Drawbacks:
  - no negative feedback
  - no preference level
```

- What are the benefits and drawbacks for explicit ratings?

```
- Benefits:
  - balanced pos and neg feedbacks
  - can have preference level
  - clear and direct

- Drawbacks:
  - biased data
  - sparse data
  - difficult to get
```

- Suppose we want to build a recommendation system based on clicks and the data of clicks are listed as follows,

```
user      item
0         1
0         0
1         1
1         4
2         3
2         2
3         3
```

What is the Utility matrix we can use for this task?

Solution:

$$\text{Utility Matrix} = \begin{bmatrix} 1 & 1 & & & \\ & 1 & & & 1 \\ & & 1 & 1 & \\ & & & 1 & \end{bmatrix}$$

- Following the last question, what is the utility matrix if we fill missing as negatives?

$$\textbf{Utility Matrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- Following the last question, what is the utility matrix if we use negative samples? (Just give one possible answer)

$$\textbf{Utility Matrix} = \begin{bmatrix} 1 & 1 & & 0 & \\ 0 & 1 & 0 & & 1 \\ & 0 & 1 & 1 & 0 \\ 0 & & 0 & 1 & \end{bmatrix}$$

- Embedded user ratings are implicit feedbacks.

```
False
```

- What is the difference between model-based filtering and memory-based filtering?

Answer:

```
- Memory based: Remember the utility matrix and make recommendations based on the KNN algorithm. It tends to be slow.
- Model based: Fit a model and make recommendations based on model predictions. It tends to have better performance. This is usually called matrix factorization.
```

- Suppose we want to predict the rating of user Alice on a movie by memory based collaborative filtering. Then what methods can we use?

```
- User-based KNN filtering: find users the nearnest to Alice, then make predictions based on the average of the ratings on that movie.
- Item-based KNN filtering: find the ratings of movies similar to that movie and then predict the average ratings based on these movies.
```

- What is the cold start problem?

Answer: If you have a new user or a new video, there will not be much signals or information to train the neural networks.

- What are the solutions for the user cold start problem?

Answer: recommend popular, require interests information, link to social media.

- How can we recommend based on the content of documents? What are the steps?

Answer: use TF-IDF. The steps are

```
- Remove stopwords
- Compute TF-IDF scores
- Keep works with high score
- Make a vector of size N to represent the document
```

- Given the following two users' ratings on items, calculate the Jaccard distance between them.

```
user 1        4 5   5 1   3 2
user 2          3 4 3 1 2 1
```

Solution:

```
sim = 1/8
```

- Given the following two users' ratings on items, and treating rating 1 and 2 to 0 and 3, 4, 5 to 1. Then calculate the Jaccard distance between them.

```
user 1        4 5   5 1   3 2
user 2          3 4 3 1 2 1
```

Solution:

```
user 1        1 1   1 0   1 0
user 2          1 1 1 0 0 0
```

If we fill missing with negatives,

```
user 1        1 1 0 1 0 0 1 0
user 2        0 1 1 1 0 0 0 0
```

Then,

```
sim = 5/8
```

- Given the following two users' ratings on items, calculate the cosine similarity between them.

```
user 1        1 5 3
user 2        1 2 3
```

Solution:

```
cosine sim = (1 + 10 + 9) / (sqrt(1 + 25 + 9) * sqrt(1 + 4 + 9))
           = 20 / (sqrt(35) * sqrt(14))
           = 20 / (7 * sqrt(10))
```

- Given the following two users' ratings on items, calculate the pearson similarity between them.

```
user 1        1 5 3
user 2        1 2 3
```

Solution:

```
avg_u1 = 3
avg_u2 = 2
pearson sim = (2 + 0 + 0) / (sqrt(4 + 4 + 0) + sqrt(1 + 0 + 1))
            = 2 / (3 * sqrt(2))
```

- What are the steps for content based kNN?

```
- Compute profile vectors for users and items
- Find a similarity measure and compute similarity between users and items
- Recommend to a user items with high similarity by kNN
```

- What are the benefits and drawbacks for a content based system?

```
Benefits:
  - no need data on other users
  - don't have a cold start problem
  - can be explained

Drawbacks:
  - difficult to construct feature vector
  - difficult to recognize new genres
  - hard to exploit the qualify of judgements from other users
```

- List two features we can use to build a user profile for movie recommendations.

Answer: Demographics, interests, search history, current location, item set, etc.

- What are the three types of negative sampling?

Answer: User-oriented sampling, item-oriented sampling, uniform random sampling

1.2 Programming

1.2.1 Predictions

Suppose we are given the embedding matrices `emb_user` and `emb_movie` from fitting the matrix factorization problem and the model is,

```
y_hat = U * V.T
```

We also have a tabular dataset `df` which has the data as follows,

userId	movieId	rating
0	0	4
0	1	3
1	1	5
1	2	5

Please write a function in `numpy` (alias `np`) and `pandas` (alias `pd`) to compute the prediction column without loops.

Solution:

```
df['prediction'] =
np.sum(emb_user[df['userId'].values]*emb_movie[df['moiveId'].values], axis=1)
```

1.2.2 Predictions

Suppose we are given the embedding matrices `emb_user` , `emb_movie` , `emb_user_bias` , `emb_movie_bias` from fitting the matrix factorization problem and the model is,

```
y_hat = U * V.T + B_U + B_V
```

We also have a tabular dataset `df` which has the data as follows,

userId	movieId	rating
0	0	4
0	1	3
1	1	5
1	2	5

Please write a function in `numpy` (alias `np`) and `pandas` (alias `pd`) to compute the prediction column without loops.

Solution:

```
df['prediction'] =
np.sum(emb_user[df['userId'].values]*emb_movie[df['moiveId'].values], axis=1)
+ emb_user_bias[df['userId'].values] + emb_movie_bias[df['moiveId'].values]
```

1.2.3 Gradients

Suppose we are using MSE for gradient descent and we are given the embedding matrices `emb_user` and `emb_movie` from fitting the matrix factorization problem.

We also have a tabular dataset `df` which has the data as follows,

userId	movieId	rating
0	0	4
0	1	3
1	1	5
1	2	5

`Y` is a sparse representation of `df`, and we are given `Y_hat` which is a sparse matrix of the prediction if the items appears in `df`.

Please write a function in `numpy` (alias `np`) and `pandas` (alias `pd`) to compute the gradients without loops.

Solution:

```
grad_user = -2 / N * np.dot((Y.toarray() - Y_hat.toarray()), emb_movie)
grad_movie = -2 / N * np.dot((Y.toarray() - Y_hat.toarray()), emb_user)
```

1.2.4 Gradient Descents

Suppose we are using momentum for calculating gradient descents and we are given the embedding matrices `emb_user` and `emb_movie` from fitting the matrix factorization problem. The dataset `df` and its sparse representation `Y` are provided.

There are also some other variables we have defined,

- `v_user` : the history velocity of user embeddings for momentum gradient descent
- `v_movie` : the history velocity of movie embeddings for momentum gradient descent
- `beta` : the weight for historical velocity
- `iterations` : the numer of iterations for gradient descent
- `learning_rate` : the given learning rate

The gradient of user and movie and be calculated through calling `gradient` within each iteration.

```
def gradient_descent(df, Y, emb_user, emb_movie):

    v_user = np.zeros_like(emb_user)
    v_movie = np.zeros_like(emb_movie)
    beta = 0.9
    iterations=100
    learning_rate=0.01

    for i in range(iterations):
        grad_user, grad_movie = gradient(df, Y, emb_user, emb_movie)
        # TODO: Implement here.


    return emb_user, emb_movie
```

Please implement the function in `numpy` (alias `np`) and `pandas` (alias `pd`) to compute the gradient descents.

Solution:

```
v_user = beta * v_user + (1 - beta) * grad_user
v_movie = beta * v_movie + (1 - beta) * grad_movie

emb_user -= learning_rate * v_user
emb_movie -= learning_rate * v_movie
```

2 Basic PyTorch

2.1 Non-Programming Question

- What is the shape of the tensor `out`?

```
embed = nn.Embedding(5,7)
x = torch.LongTensor([[1,0,1,4,2]])
out = embed(x)
```

Answer:

```
[1, 5, 7]
```

- What is the minimum valid integer for the blank?

```
embed = nn.Embedding(_______,7)
x = torch.LongTensor([[1,1,2,2,3],[5,4,4,2,1]])
out = embed(x)
```

Answer:

```
6
```

- What is the shape of the tensor `out`?

```
embed = nn.Embedding(10,5)
x = torch.LongTensor([1,0,1,4,2,0])
out = embed(x)
```

Answer:

```
[6, 5]
```

- What is the value of `result` in the last line?

```
embed = nn.Embedding(10,5)
x = torch.LongTensor([1,0,1,4,2,1])
out = embed(x)
bools = out[0] == out[2]
result = bools.detach().numpy()[0]
```

Answer:

```
True
```

- What is the value of the tensor *x.grad* after running the next few lines?

```
x = torch.tensor([1.0,3.0,2.0], requires_grad=True)
L = (3*x + 7).sum()
L.backward()
```

Answer:

```
tensor([3., 3., 3.])
```

- What is the value of the tensor *x.grad* after running the next few lines?

```
x = torch.tensor([1.0,3.0,2.0], requires_grad=True)
L = (-2*x**2 + 3*x + 7).mean()
L.backward()
```

Answer:

```
tensor([-0.3333, -3.0000, -1.6667])
```

- How to create a tensor of zeros shaped `[2,2,2]`?

```
torch.zeros(2,2,2)
```

- How to create a tensor of ones shaped `[2,2,2]`?

```
torch.ones(2,2,2)
```

- How to create a tensor of normal distributed random values ranged [0, 1) of shape `[2,2,2]`?

```
torch.rand(2,2,2)
```

- How to create a tensor of uniform distributed random integers ranged [3, 10) of shape `[2,2,2]`?

```
torch.randint(3, 10, (2,2,2))
```

- What is the shape of the `x` after the following lines?

```
x = torch.tensor([[1,2,3,4]])
x = x.squeeze()
```

Answer:

```
[4]
```

- What is the shape of the `x` after the following lines?

```
x = torch.tensor([[1,2,3,4]])
x = x.squeeze(1)
```

Answer:

```
[1, 4]
```

- What is the shape of the `x` after the following lines?

```
x = torch.tensor([[1],[2],[3],[4]])
x = x.squeeze()
```

Answer:

```
[4]
```

- What is the shape of the `x` after the following lines?

```
x = torch.tensor([[1,2,3,4]])
x = x.unsqueeze(1)
```

Answer:

```
[1,1,4]
```

- What is the shape of the `x` after the following lines?

```
x = torch.tensor([[1],[2],[3],[4]])
x = x.unsqueeze(1)
```

Answer:

```
[4,1,1]
```

- What is the shape of the `x` after the following lines?

```
x = torch.tensor([1,2,3,4])
x = x.unsqueeze(1)
```

Answer:

```
[4,1]
```

- Describe what each of the following lines of code are doing during a training loop.

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

Answer:

```
line 1: zero out the gradients for all the tensors in the model
line 2: computing the gradients for all tensors based on loss
line 3: iterate all thee tensors and use the interally stored grad to update
their values
```

- Suppose we are given batch size of 1,000 and the data size is 1,000,000, then calculate how many batches do we have in one epoch.

```
1000000/1000 = 1000
```

- Suppose we are given batch size of 1,000 and the data size is 1,000,000. The total number of epoches is 10 per training. Calculate the number of iterations we have in one training.

```
1000000/1000 * 10 = 10000
```

2.2 Programming Question

2.2.1 PyTorch Model

Suppose we have a linear regression problem with 4 input variables. Write a model in PyTorch that can be used for this task. Suppose we have imported,

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

Use the following template for your implementation.

```
class LinearRegression(torch.nn.Module):

    def __init__(self):
        super(LinearRegression, self).__init__()
        # TODO: Implement this method

    def forward(self, x):
        # TODO: Implement this method

model = LinearRegression()
```

Solution:

```
class LinearRegression(torch.nn.Module):

    def __init__(self):
        super(LinearRegression, self).__init__()
        self.linear = torch.nn.Linear(4, 1)

    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred

model = LinearRegression()
```

2.2.2 PyTorch Model

Suppose we have a logistic regression problem with 4 input variables. Write a model in PyTorch that can be used for this task. Suppose we have imported,

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

Use the following template for your implementation.

```
class LogisticLinearRegression(torch.nn.Module):

    def __init__(self):
        super(LinearRegression, self).__init__()
        # TODO: Implement this method

    def forward(self, x):
        # TODO: Implement this method

model = LogisticLinearRegression()
```

Solution:

```
class LogisticRegression(torch.nn.Module):

     def __init__(self):
        super(LogisticRegression, self).__init__()
        self.linear = torch.nn.Linear(4, 1)

     def forward(self, x):
        y_pred = F.sigmoid(self.linear(x))
        return y_pred

model = LogisticRegression()
```

2.2.3 PyTorch Model

Suppose we have a classification problem with 10 input variables and 3 output classes. Write a model in PyTorch that can be used for this task. Suppose we have imported,

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

Use the following template for your implementation.

```
class Classification(torch.nn.Module):

    def __init__(self):
        super(Classification, self).__init__()
        # TODO: Implement this method

    def forward(self, x):
        # TODO: Implement this method

model = Classification()
```

Solution:

```
class Classification(torch.nn.Module):

    def __init__(self):
        super(Classification, self).__init__()
        self.linear = torch.nn.Linear(10, 3)
        self.softmax = torch.nn.Softmax(dim=1)

    def forward(self, x):
        y_pred = self.softmax(self.linear(x))
        return y_pred

model = Classification()
```

2.2.4 PyTorch Model

Suppose we have a matrix factorization problem with embedding size of 100 without bias. Write a model in PyTorch that can be used for this task. Suppose we have imported,

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

Use the following template for your implementation.

```
class MF(nn.Module):
    def __init__(self, num_users, num_items, emb_size=100):
        super(MF, self).__init__()
        # TODO: Implement this method

    def forward(self, u, v):
        # TODO: Implement this method

model = MF()
```

Solution:

```
class MF(nn.Module):
    def __init__(self, num_users, num_items, emb_size=100):
        super(MF, self).__init__()
        self.user_emb = nn.Embedding(num_users, emb_size)
```

```
        self.item_emb = nn.Embedding(num_items, emb_size)
        self.user_emb.weight.data.uniform_(0, 0.05)
        self.item_emb.weight.data.uniform_(0, 0.05)

    def forward(self, u, v):
        U = self.user_emb(u)
        V = self.item_emb(v)
        return (U * V).sum(1)

model = MF()
```

2.2.5 PyTorch Model

Suppose we have a matrix factorization problem with embedding size of 100 with bias. Write a model in PyTorch that can be used for this task. Suppose we have imported,

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

Use the following template for your implementation.

```
class BiasedMF(nn.Module):
    def __init__(self, num_users, num_items, emb_size=100):
        super(BiasedMF, self).__init__()
        # TODO: Implement this method

    def forward(self, u, v):
        # TODO: Implement this method

model = BiasedMF()
```

Solution:

```
class BiasedMF(nn.Module):
    def __init__(self, num_users, num_items, emb_size=100):
        super(BiasedMF, self).__init__()

        self.user_emb = nn.Embedding(num_users, emb_size)
        self.item_emb = nn.Embedding(num_items, emb_size)
```

```
        self.user_bias = nn.Embedding(num_users, 1)
        self.item_bias = nn.Embedding(num_item, 1)

        self.user_emb.weight.data.uniform_(0, 0.05)
        self.item_emb.weight.data.uniform_(0, 0.05)
        self.user_bias.weight.data.uniform_(-0.01, 0.01)
        self.item_bias.weight.data.uniform_(-0.01, 0.01)

    def forward(self, u, v):
        U = self.user_emb(u)
        V = self.item_emb(v)
        b_u = self.user_bias(u).squeeze()
        b_v = self.item_bias(v).squeeze()
        return (U * V).sum(1) + b_u + b_v

model = BiasedMF()
```

2.2.6 Train loop

Write a training loop in PyTorch that can train a linear regression model. The template is given as follows,

```
def train_model(model, optimizer, train_dl, epoch=10):
    for i in range(epochs):
        model.train()
        total = 0
        sum_loss = 0
        for x, y in train_dl:
            batch = y.shape[0]
            # TODO: Implement here.

            total += batch
            sum_loss += batch*(loss.item())
        train_loss = sum_loss / total
        print(train_loss)
```

Solution:

```
y_hat = model(x)
loss = F.mse_loss(y_hat, y)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

2.2.7 Train loop

Write a training loop in PyTorch that can train a logistic regression model (assume the model is linear and doesn't apply sigmoid in the end). The template is given as follows,

```
def train_model(model, optimizer, train_dl, epoch=10):
    for i in range(epochs):
        model.train()
        total = 0
        sum_loss = 0
        for x, y in train_dl:
            batch = y.shape[0]
            # TODO: Implement here.

            total += batch
            sum_loss += batch*(loss.item())
        train_loss = sum_loss / total
        print(train_loss)
```

Solution:

```
y_hat = model(x)
loss = F.binary_cross_entropy_with_logit(y_hat, y)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

2.2.8 Train loop

Write a training loop in PyTorch that can train a multiclass classification model. The template is given as follows,

```
def train_model(model, optimizer, train_dl, epoch=10):
    for i in range(epochs):
        model.train()
        total = 0
        sum_loss = 0
        for x, y in train_dl:
            batch = y.shape[0]
            # TODO: Implement here.

            total += batch
            sum_loss += batch*(loss.item())
        train_loss = sum_loss / total
        print(train_loss)
```

Solution:

```
y_hat = model(x)
loss = F.cross_entropy(y_hat, y)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

2.2.9 Dataset & Dataloader

Suppose we are given `x_train` and `y_train` , construct a dataloader `train_dl` with batch size of 1000, and shuffle the data.

Solution:

```
class TrainingDataset(data_utils.Dataset):

    def __init__(self, x, y):
        x = torch.tensor(x).float()
        y = torch.tensor(y).float().unsqueeze(1)
        self.x, self.y = x, y

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

    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]

```

```
train_ds = TrainingDataset(x_train, y_train)
train_dl = data_utils.DataLoader(train_ds, batch_size=1000, shuffle=True)
```

2.2.10 Dataloader

Suppose we want to get one batch of data from the dataloader `train_dl`. Please give the function for this task.

Solution:

```
x, y = next(iter(train_dl))
```

2.2.11 Dataloader

Write a function that given a model `model` and a dataloader `train_dl` that computes the balanced accuracy. Assume that you have a binary classification problem and you can use `balanced_accuracy_score` from `sklearn`.

Solution:

```
def metric_accuracy(model, dataload):

    model.eval()
    accuracies = []

    for x, y in dataload:

        y_hat = model(x)
        accuracy = sklearn.metrics.balanced_accuracy_score(y, y_hat)
        accuracies.append(accuracy)

    return np.mean(accuracies)

metric_accuracy(model, train_dl)
```

3 AdaBoosting

3.1Non-Programming Question

- What are the features of a bushy tree?

Answer:

```
Tends to overfit, high variance, low bias
```

- What are the features of a shallow tree?

Answer:

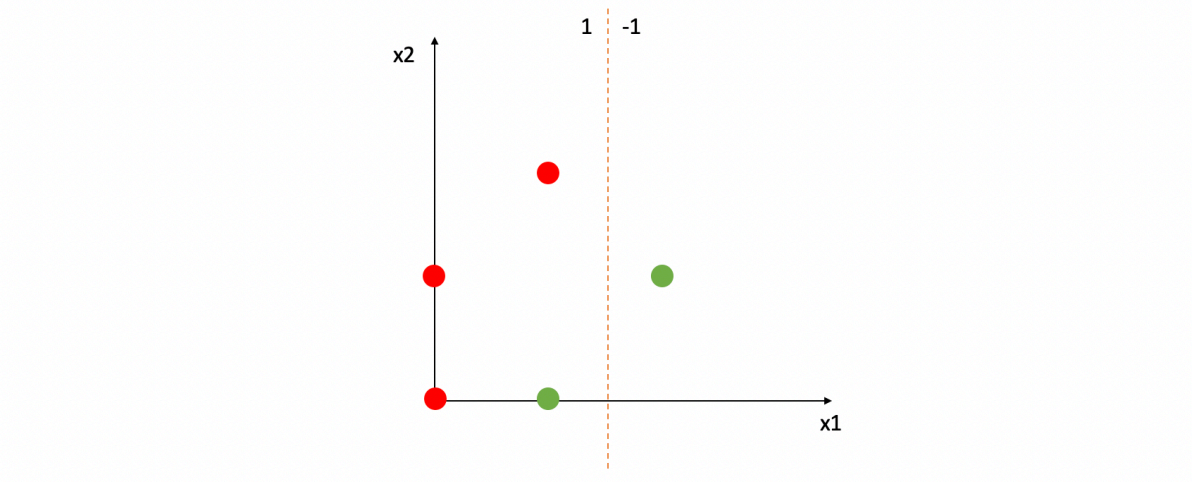
```
Tends to underfit, high bias.
```

- Suppose we have the following dataset. What is the best split for a minimized misclassification error? And report this misclassification error. Let's suppose all the points have the same weight.

X1	X2	y
0	0	1
0	1	1
1	2	1
1	0	-1
2	1	-1

Solution:

We can drow a plot as follows.



From this plot, we can know the best split can be $x1 < 1.5$. And the misclassification error is `0.2`.

- Continue with the last question, what are the Gini impurity and the cross entropy for the classification after that split?

Answer:

$$Gini = 1/5 * (1 - 1/5) + 4/5 * (1 - 4/5) = 0.32$$

$$Entropy = -(1/5 * log_2(1/5) + 4/5 * log_2(4/5)) = 0.72$$

- Suppose we have the following decision tree. Then how many terminal regions do we have finally?

```
Split 1: x1 > 1.3
Split 2: x2 <= 0.45
```

Answer:

```
4, because we have two splits on different features
```

- Suppose we have the following decision tree. Then how many terminal regions do we have finally?

```
Split 1: x1 > 1.3
Split 2: x1 <= 0.45
```

Answer:

```
3, because we have two splits on the same feature
```

- Suppose we have a decision tree as follows.

$$T(x;\theta) = \sum_{j=1}^{J} \beta_j 1_{[x \in R_j]}$$

Then how many parameters do we have for this tree? And how many regions do we have as a result?

Answer:

```
We have 2J parameters (R1 ... RJ and b1 ... bJ).
We have J regions (R1 ... RJ).
```

- Give three examples of the weak classifiers.

Answer:

```
logistic regression, shallow decision tree, decision stumps
```

- What are the features of the weak classifiers?

Answer:

```
- fast to fit
- don't overfit
- tend to underfit
- high bias
- do not have good accuracy
```

- How to get a strong classifier? (Give three examples)

Answer:

```
- add more features or add polynomial features
- use an ensemble model with bagging
- combine weak classifiers
```

- What is the prediction in the following example?

Let's suppose the following information is given,

$$x = (120K, Bad, 50K, Good)$$

And the weights for each tree in the model are,

$$\alpha_1 = 2,\ \alpha_2 = 1.5,\ \alpha_3 = 1.5,\ \alpha_4 = 0.5$$

Solution:

Let's suppose the labels are $safe = 1$ and $Risky = -1$. Then,

$$T_1(x) = 1$$
$$T_2(x) = -1$$
$$T_3(x) = -1$$
$$T_4(x) = 1$$

So the hard prediction is,

$$f_4(x) = sign(\sum_{m=1}^{4} \alpha_m T_m(x)) = sign(2 - 1.5 - 1.5 + 0.5) = -1$$

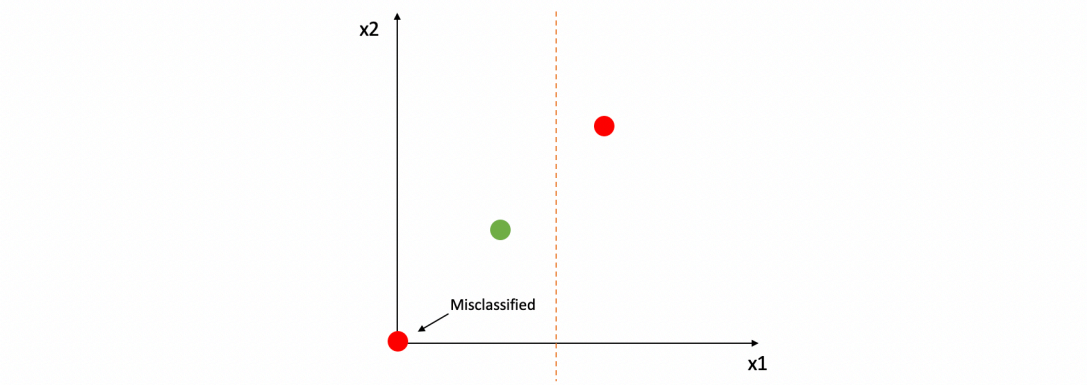
So the prediction for this case is the label `Risky`.

- Consider we have the following weighted dataset.

x1	X2	y	w
0	0	1	1/5
1	1	-1	3/5
2	2	1	2/5

Find a stump that minimize the weighted classification error.

Solution: We can draw the following plot.



So that the stump with $x1 > 1.5$ will have a minimized misclassified error of 1/6.

Another possible answer can be $x2 > 1.5$ which has the same misclassified error.

- Consider the following dataset.

x1	x2	y	w
0	0	-1	2/10
0	1	-1	2/10
1	1	-1	3/10
1	3	1	1/10
2	1	1	2/10

Compute the weighted classification error of the following classifier. Which points are misclassified?

$$g(x) = \begin{cases} 1, & \text{if } x_2 \geq 0.5 \\ -1, & \text{otherwise} \end{cases}$$

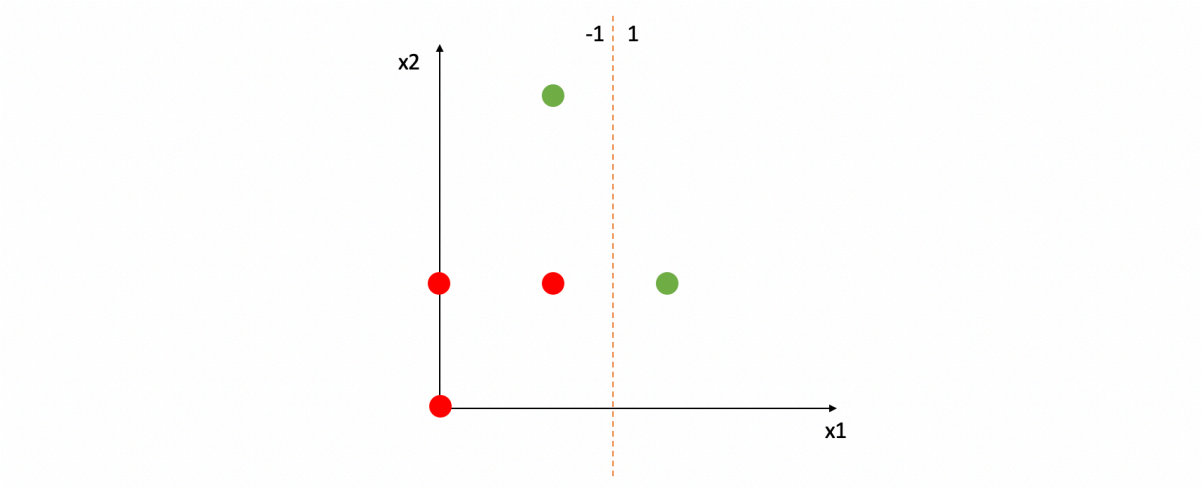
Answer:

Point 2 and 3 are misclassified. The weighted misclassification error is,

$$error = (2+3)/(2+2+3+1+2) = 1/2$$

- Continue with the last problem, find a stump that minimizes the weighted misclassification error and report this error.

Solution: We can draw a plot as follows.



Then the stump with $x1 > 1.5$ minimizes the error. The error in this case is $1/10$.

- Consider the following dataset and `play tennis` is the label.

Tempature	Wind	play tennis
hot	F	no
hot	T	no
hot	F	yes
mild	F	yes
cool	F	yes
cool	T	no

Run Adaboosting by hand for 2 iterations.

Solution:

X1	X2	Y	W0
0	0	-1	1/6
0	1	-1	1/6
0	0	1	1/6
1	0	1	1/6
2	0	1	1/6
2	1	-1	1/6

The first stump should be $x2 < 0.5$ with error of `1/6`. So $\alpha_1 = log(5)$ and the first point is misclassified.

X1	X2	Y	W0	T1	W1
0	0	-1	1/6	1	5/6
0	1	-1	1/6	-1	1/6
0	0	1	1/6	1	1/6
1	0	1	1/6	1	1/6
2	0	1	1/6		1/6
2	1	-1	1/6	-1	1/6

The second stump should be $x1 > 0.5$ with error of,

$$error = (1 + 1)/(5 + 5 * 1) = 1/5$$

So $\alpha_2 = log(4)$ and the point 3, 6 is misclassified.

X1	X2	Y	W0	T1	W1	T2	W2
0	0	-1	1/6	1	5/6	-1	5/6
0	1	-1	1/6	-1	1/6	-1	1/6
0	0	1	1/6	1	1/6	-1	4/6
1	0	1	1/6	1	1/6	1	1/6
2	0	1	1/6	1	1/6	1	1/6
2	1	-1	1/6	-1	1/6	1	4/6

The final predictions are,

$$\hat{y}_1 = sign(log5 - log4) = 1$$
$$\hat{y}_1 = sign(-log5 - log4) = -1$$
$$\hat{y}_1 = sign(log5 - log4) = 1$$
$$\hat{y}_1 = sign(log5 + log4) = 1$$
$$\hat{y}_1 = sign(log5 + log4) = 1$$
$$\hat{y}_1 = sign(-log5 + log4) = -1$$

3.2 Pseudo Code for AdaBoosting

- Fill in the following blanks.

Algorithm 2 Adaboost

1: **procedure** ADABOOT
2: Initialize weights _____1_____
3: **for** $m = 1$ to M **do**
4: Fit a tree classifier $T_m(x) = T(x; \theta_m)$ to training data using weights w_i
5: Compute

$$err_m = \text{\_\_\_\_\_2\_\_\_\_\_}$$

6: Compute $\alpha_m =$ _____3_____
7: Set $w_i =$ _____4_____
8: **end for**
9: _____5_____
10: **end procedure**

Solution:

1.

$$w_i = \frac{1}{N}$$

2.

$$\frac{\sum_{i=1}^{N} w_i 1[y_i \neq T_m(x_i)]}{\sum_{i=1}^{N} w_i}$$

3.

$$\log \frac{1 - err_m}{err_m}$$

4.

$$w_i \cdot e^{\alpha_m 1[y_i \neq T_m(x_i)]}$$

5.

$$F(x) = sign(\sum_{m=1}^{M} \alpha_m T_m(x))$$

4 Gradient Boosting

4.1 Non-Programming Questions

- What's the meaning of forward stepwise additive model?

Answer:

```
- forward: means we don't go back
- stepwise: means we take one step for each iteration
- additive: means we sum up the models to get a prediction
```

- Loss functions for binary classification are written as a function of margin.

```
True
```

- Hinge loss is also called SVM loss.

```
True
```

- What is the difference between adaboosting and gradient boosting?

```
- trees in adaboosting are fitted on the same data changing weights
- trees in gradient boosting are fitted by pseudo residual which changes every iteration
```

- Gradient boosting is a generalized Adaboosting.

```
True
```

- What is the difference between gradient boosting and random forest?

```
- random forest uses bagging and grow fully grown decision trees, while gradient boosting use weak learners like shallow trees.
- the initial bias for random forest is made as low as possible, but gradient boosting reduces the bias by adding more trees.
```

- What is the difference between gradient boosting and neural network?

```
- gradient boosting works well on tabular data
- neural networks are useful when applied to raw sensory data
```

- Suppose we have the following data.

Sqfeet	**Rent**
750	1160
800	1200
850	1280
900	1450
950	2000

Compute by hand with gradient boosting for two iterations with MAE loss.

Solution:

The first step is,

x	y	f0	y-f0	Pr0
750	1160	1280	-120	-1
800	1200	1280	-80	-1
850	1280	1280	0	0
900	1450	1280	170	1
950	2000	1280	720	1

To fit a tree based on pseudo residual 0, we have to split at $x = 825$. Then we get the median as our prediction for the pseudo residual, and construct $f1$.

x	y	f0	y-f0	Pr0	T1	f1 = f0 + T1
750	1160	1280	-120	-1	-100	1180
800	1200	1280	-80	-1	-100	1180
850	1280	1280	0	0	170	1450
900	1450	1280	170	1	170	1450
950	2000	1280	720	1	170	1450

Continue this process,

x	y	f0	y-f0	Pr0	T1	f1	y-f1	Pr1
750	1160	1280	-120	-1	-100	1180	-20	-1
800	1200	1280	-80	-1	-100	1180	20	1
850	1280	1280	0	0	170	1450	-170	-1
900	1450	1280	170	1	170	1450	0	0
950	2000	1280	720	1	170	1450	550	1

To fit a tree, we have to split on $x = 775$, and then take the median for T_2,

x	y	f0	y-f0	Pr0	T1	f1	y-f1	Pr1	T2
750	1160	1280	-120	-1	-100	1180	-20	-1	-20
800	1200	1280	-80	-1	-100	1180	20	1	10
850	1280	1280	0	0	170	1450	-170	-1	10
900	1450	1280	170	1	170	1450	0	0	10
950	2000	1280	720	1	170	1450	550	1	10

Then by $f_2 = f_1 + T_2$, we have,

x	y	f0	y-f0	Pr0	T1	f1	y-f1	Pr1	T2	f2
750	1160	1280	-120	-1	-100	1180	-20	-1	-20	1160
800	1200	1280	-80	-1	-100	1180	20	1	10	1190
850	1280	1280	0	0	170	1450	-170	-1	10	1460
900	1450	1280	170	1	170	1450	0	0	10	1460
950	2000	1280	720	1	170	1450	550	1	10	1460

And f_2 is our prediction after two iterations of gradient boosting with MAE loss.

- Suppose we have the following data.

Sqfeet	Rent
750	1160
800	1200
850	1280
900	1450
950	2000

Compute by hand with gradient boosting for one iteration with MSE loss.

Solution:

Because we have MSE loss, the best initial model is to take the mean value of y.

x	y	f0	Pr0 = y-f0
750	1160	1418	-258
800	1200	1418	-218
850	1280	1418	-138
900	1450	1418	32
950	2000	1418	582

To minimize MSE loss, we have to fit a stump that split at $x = 925$. So, then we take the mean of the pseudo residual in the same split as predictions

x	y	f0	Pr0	T1	f1=f0+T1
750	1160	1418	-258	-145.5	1272.5
800	1200	1418	-218	-145.5	1272.5
850	1280	1418	-138	-145.5	1272.5
900	1450	1418	32	-145.5	1272.5
950	2000	1418	582	582	2000

And f_1 is our prediction after one iteration of gradient boosting with MSE loss.

- Suppose we have the following data.

Sqfeet	Rent
750	1160
800	1200
850	1280
900	1450
950	2000

Compute by hand with gradient boosting for one iteration with MSE loss. Note that we have a shrinkage (learning rate) of 0.8 in this case.

Solution:

This question is similar to the question above and what we have to change is to mutiply that shrinkage when calculating f_1. So,

x	y	f0	Pr0	T1	νT1	f1=f0+νT1
750	1160	1418	-258	-145.5	-116.4	1301.6
800	1200	1418	-218	-145.5	-116.4	1301.6
850	1280	1418	-138	-145.5	-116.4	1301.6
900	1450	1418	32	-145.5	-116.4	1301.6
950	2000	1418	582	582	465.6	1883.6

Here, f_1 should be our prediction after one iteration of gradient boosting with MSE loss and shrinkage of 0.8.

- Suppose we have a learning rate for gradient boosting as η and we find an optimized iteration number M. Then if we want to devide the learning rate by 2 for finding the best learning rate, how should we change M?

Answer:

```
Increase M to 2M
```

- Suppose we have the loss function for gradient boosting as,

$$L(y, f(x)) = \log(1 + e^{-yf(x)})$$

Then calculate the best initial constant for its initial model.

Answer:

So the best constant α should be found by,

$$\alpha = \arg\min_{\alpha} \sum_{i=1}^{N} \log(1 + e^{-y^{(i)}\alpha})$$

Then,

$$\frac{d\sum_{i=1}^{N}\log(1+e^{-y^{(i)}\alpha})}{d\alpha}=-\sum_{i=1}^{N}\frac{1}{1+e^{-y^{(i)}\alpha}}y^{(i)}e^{-y^{(i)}\alpha}=0$$

So,

$$\sum_{i=1}^{N}\frac{y^{(i)}}{1+e^{y^{(i)}\alpha}}=0$$

Because $y^{(i)}\in\{1,-1\}$, and we can define N^+ as the number of $y^{(i)}=1$ and N^- as the number of $y^{(i)}=-1$. So we have,

$$N=N^++N^-$$

Therefore,

$$\sum_{i=1}^{N}\frac{y^{(i)}}{1+e^{y^{(i)}\alpha}}=\frac{N^+}{1+e^{\alpha}}-\frac{N^-}{1+e^{-\alpha}}=0$$

So,

$$\frac{N^+}{1+e^{\alpha}}=\frac{N^-}{1+e^{-\alpha}}$$

Then,

$$\frac{N^+}{1+e^{\alpha}}=\frac{N^-e^{\alpha}}{1+e^{\alpha}}$$

So,

$$N^+=N^-e^{\alpha}$$

And,

$$\alpha=\log\frac{N^+}{N^-}$$

Note that we have,

$$\frac{1+\bar{y}}{1-\bar{y}}=\frac{1+\frac{N^+-N^-}{N}}{1-\frac{N^+-N^-}{N}}=\frac{N+N^+-N^-}{N-N^++N^-}=\frac{N^+}{N^-}$$

So α is also,

$$\alpha=\log\frac{1+\bar{y}}{1-\bar{y}}$$

- Calculate the pseudo residual for MSE loss.

Solution:

For MSE, the pseudo residual is quite simple with,

$$MSE = (y - f)^2$$

Then,

$$r_{im} = -\frac{\partial (y-f)^2}{\partial f}\bigg|_{f=f_{m-1}(x^{(i)}),\ y^{(i)}} = -2(y^{(i)} - f_{m-1}(x^{(i)}))$$

- Calculate the pseudo residual for MAE loss.

Solution:

For MAE, the loss function is,

$$MAE = |y - f|$$

Then,

$$r_{im} = -\frac{\partial |y-f|}{\partial f}\bigg|_{f=f_{m-1}(x^{(i)}),\ y^{(i)}}$$

This is to say we have,

$$r_{im} = \begin{cases} -\frac{\partial (y-f)}{\partial f}\Big|_{f=f_{m-1}(x^{(i)}),\ y^{(i)}}, & y - f \geq 0 \\ -\frac{\partial (f-y)}{\partial f}\Big|_{f=f_{m-1}(x^{(i)}),\ y^{(i)}}, & y - f < 0 \end{cases}$$

Then,

$$r_{im} = \begin{cases} 1, & y - f \geq 0 \\ -1, & y - f < 0 \end{cases}$$

So that we can conclude that, for MAE, the pseudo result can be written as,

$$r_{im} = sign(y^{(i)} - f_{m-1}(x^{(i)}))$$

- Calculate the pseudo residual for the following loss function,

$$L(y, f) = e^{-yf}$$

Solution:

For the definition of pseudo residual,

$$r_{im} = -\frac{\partial e^{-yf}}{\partial f}\bigg|_{f=f_{m-1}(x^{(i)}),\ y^{(i)}} = y^{(i)} \cdot e^{-y^{(i)} f_{m-1}(x^{(i)})}$$

- Calculate the pseudo residual for the following loss function,

$$L(y, f) = \max(0, 1 - yf)$$

Solution:

The loss function can also be written as,

$$L(y, f) = \begin{cases} 0, & yf \geq 1 \\ 1 - yf, & yf < 1 \end{cases}$$

Then the gradient of L to f is,

$$r_{im} = \begin{cases} 0, & yf \geq 1 \\ -\frac{\partial(1-yf)}{\partial f}\Big|_{f=f_{m-1}(x^{(i)}),\ y^{(i)}}, & yf < 1 \end{cases}$$

So we have,

$$r_{im} = \begin{cases} 0, & yf \geq 1 \\ y^{(i)}, & yf < 1 \end{cases}$$

- Calculate the pseudo residual for the following loss function,

$$L(y, f) = \log(1 + e^{-yf})$$

Solution:

Then,

$$r_{im} = -\frac{\partial \log(1 + e^{-yf})}{\partial f}\bigg|_{f=f_{m-1}(x^{(i)}),\ y^{(i)}}$$

This is also,

$$r_{im} = -\frac{-y \cdot e^{-yf}}{\log(1 + e^{-yf})}\bigg|_{f=f_{m-1}(x^{(i)}),\ y^{(i)}}$$

So,

$$r_{im} = -\frac{-y}{\log\left(1+e^{yf}\right)}\bigg|_{f=f_{m-1}\left(x^{(i)}\right),\ y^{(i)}} = \frac{y^{(i)}}{\log\left(1+e^{y^{(i)}f_{m-1}\left(x^{(i)}\right)}\right)}$$

- Calculate the best constant per region for logloss.

Solution:

After getting the pseudo residual and fitting the model , then, the next step is to find the optimal constant per region. For logloss, that is,

$$\begin{aligned}\beta_j &= \arg\min_{\beta} \sum_{x^{(i)}\in R_j} L\left(y^{(i)}, f_{m-1}\left(x^{(i)}\right)+\beta\right) \\ &= \arg\min_{\beta} \sum_{x^{(i)}\in R_j} \log\left(1+e^{-y^{(i)}\left[f_{m-1}\left(x^{(i)}\right)+\beta\right]}\right)\end{aligned}$$

Then the derivative is,

$$G(\beta) = \frac{\partial \sum_{x^{(i)}\in R_j} \log\left(1+e^{-y^{(i)}\left[f_{m-1}\left(x^{(i)}\right)+\beta\right]}\right)}{\partial\beta} = -\sum_{x^{(i)}\in R_j} \frac{y^{(i)}}{1+e^{y^{(i)}\left[f_{m-1}\left(x^{(i)}\right)+\beta\right]}}$$

Because $G(\beta)=0$ doesn't have a closed form solution, we will use the following rule for estimating β

$$\beta = -\frac{G(0)}{G'(0)}$$

So, firstly, $G'(\beta)$ is ,

$$G'(\beta) = \sum_{x^{(i)}\in R_j} \frac{\left(y^{(i)}\right)^2 e^{y^{(i)}\left[f_{m-1}\left(x^{(i)}\right)+\beta\right]}}{\left(1+e^{y^{(i)}\left[f_{m-1}\left(x^{(i)}\right)+\beta\right]}\right)^2}$$

Note that,

$$-G(0) = \sum_{x^{(i)}\in R_j} \frac{y^{(i)}}{1+e^{y^{(i)}\left[f_{m-1}\left(x^{(i)}\right)\right]}} = \sum_{x^{(i)}\in R_j} r_i$$

And,

$$|r_i| = \frac{1}{1+e^{y^{(i)}\cdot f_{m-1}\left(x^{(i)}\right)}}$$

So,

$$1 - |r_i| = \frac{e^{y^{(i)} \cdot f_{m-1}(x^{(i)})}}{1 + e^{y^{(i)} \cdot f_{m-1}(x^{(i)})}}$$

Therefore,

$$\beta_j = \frac{\sum_{x^{(i)} \in R_j} r_i}{\sum_{x^{(i)} \in R_j} |r_i| \left(1 - |r_i|\right)}$$

4.2 Gradient Boosting Pseudo Code

- Fill in the blanks for Gradient boosting with MSE loss.

Algorithm 4 Gradient Boosting with squared-error loss

procedure GRADIENT BOOSTING FOR SQUARED-ERROR LOSS
  Initialize $f_0(x) =$ 1
  for $m = 1$ to M **do**
    (a) Compute the residual vector $r = (r_1, \ldots, r_N)$ between y and the current prediction $f_{m-1}(x)$

$$r_i = \text{2}$$

    (b) Fit a regression tree to the targets r_i giving $T_m(x)$
    (c) Update $f_m(x) =$ 3
  end for
  Return $f(x) =$ 4
end procedure

Answer:

1.

$$\bar{y}$$

2.

$$y_i - f_{m-1}(x_i)$$

3.

$$f_{m-1}(x) + T_m(x)$$

4.

$$f_M(x)$$

- Fill in the blanks for Gradient boosting with MAE loss.

Algorithm 5 Gradient Boosting with MAE

1: **procedure** GRADIENT BOOSTING FOR MAE
2: Initialize $f_0(x) =$ 1
3: **for** $m = 1$ to M **do**
4: (a) Compute the pseudo-residual vector $r = (r_1, \ldots, r_N)$ between y and the current prediction $f_{m-1}(x)$

$$r_i =$$ 2

5: (b) Fit a regression tree to the targets r_i giving terminal regions $R_{jm}, j = 1 \ldots J_m$
6: (c) Compute new predictions for every terminal node R_{jm}

$$\beta_{jm} =$$ 3

7: (d) Update $f_m(x) =$ 4
8: **end for**
9: Return $f(x) =$ 5
10: **end procedure**

Solution:

1.

$$\text{median}(y_i)$$

2.

$$\text{sign}(y_i - f_{m-1}(x_i))$$

3.

$$\text{median}_{x_i \in R_{jm}}(y_i - f_{m-1}(x_i))$$

4.

$$f_{m-1}(x) + \sum_{j=1}^{Jm} \beta_{jm} 1_{[x_i \in R_{jm}]}$$

5.

$$f_M(x)$$

- Fill in the blanks for general Gradient tree boosting.

Algorithm 1 Gradient Tree Boosting Algorithm

1: Initialize $f_0(x) =$ 1

2: **for** $m = 1$ to M **do**

3: (a) Compute negative gradient of the loss function:

$$r_i = \text{2}$$

4: (b) Fit a regression tree to the targets r_i giving terminal regions $R_{jm}, j = 1 \dots J_m$

5: (c) Compute new predictions for every terminal node.

$$\beta_{jm} = \text{3}$$

6: (d) Update $f_m(x) =$ 4

7: **end for**

8: Return $f(x) =$ 5

Solution:

1.

$$\arg\min_{\beta} \sum_{i=1}^{N} L(y_i, \beta)$$

2.

$$-\frac{\partial \log\left(1 + e^{-yf}\right)}{\partial f}\bigg|_{f=f_{m-1}(x_i),\, y_i}$$

3.

$$\arg\min_{\beta} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \beta)$$

4.

$$f_{m-1}(x) + \sum_{j=1}^{Jm} \beta_{jm} 1_{[x_i \in R_{jm}]}$$

5.

$$f_M(x)$$

- Fill in the blank for this gradient boosting with log loss.

Algorithm 6 Gradient Boosting Binary Classification with Log Loss

procedure GRADIENT BOOSTING FOR LOG LOSS
  Initialize $f_0(x) =$ 1
  for $m = 1$ to M **do**
    (a) Compute the pseudo-residual vector $r = (r_1, \ldots, r_N)$ between y and the current prediction $f_{m-1}(x)$

$$r_i = \text{2}$$

    (b) Fit a regression tree to the targets r_i giving terminal regions $R_{jm}, j = 1 \ldots J_m$
    (c) Compute new predictions for every terminal node R_{jm}

$$\beta_{jm} = \text{3}$$

    (d) Update $f_m(x) =$ 4
  end for
  Return $f(x) = f_M(x)$
end procedure

Solution:

1.

$$\log \frac{1+\bar{y}}{1-\bar{y}}$$

2.

$$\frac{y^{(i)}}{\log\left(1 + e^{y^{(i)} f_{m-1}(x^{(i)})}\right)}$$

3.

$$\frac{\sum_{x^{(i)} \in R_j} r_i}{\sum_{x^{(i)} \in R_j} |r_i| \left(1 - |r_i|\right)}$$

4.

$$f_{m-1}(x) + \sum_{j=1}^{Jm} \beta_{jm} 1_{[x_i \in R_{jm}]}$$