



# Data Intelligence

**Field-aware Factorization Machine - Lab**

**U Kang  
Seoul National University**



# In This Lecture

- Implement multiple models, including FFM, which can be used when predicting click-through rate
- Compare strengths and weaknesses of the models



# Outline

- ➡ ☐ Introduction
- ☐ Data
- ☐ Preprocessing Codes
- ☐ Models
  - ☐ Logistic Regression
  - ☐ Degree-2 Polynomial Mapping
  - ☐ Factorization Machine
  - ☐ Field-aware Factorization Machine
- ☐ Result



# Motivation

- **Predicting click-through rate (CTR)** of an individual is very helpful to improve advertisers' revenues
- If we consider feature conjunction in a field-wise latent space, in addition to individual feature weights, we will get better performance.



# Goals

- Classify click preference by learning the feature conjunction in a field-wise latent space
- Practice session
  - Predicting **income level** by demographic information in 'adult' dataset



# Problem Definition (1)

- **Given:** A bunch of demographic features such as age, education, sex, marital status, race, etc.
  - $x$  – features
  
- Learn  $P(y|x)$
  
- **Predict:** Whether he/she makes over 50K a year
  - $y$  – click probability



# Problem Definition (2)

Income(Y)	Age(X1)	Sex(X2)	Race(X3)
< 50K	19	Male	White
≥ 50K	41	Female	Black
...	...	...	...

↓ Learn

$$P(y|x)$$

↓ Predict

Income(Y)	Age(X1)	Sex(X2)	Race(X3)
?	50	Female	White
?	23	Male	Black
...	...	...	...



# Outline

☒ Introduction

 ☐ **Data**

☐ Preprocessing Codes

☐ Models

☐ Logistic Regression

☐ Degree-2 Polynomial Mapping

☐ Factorization Machine

☐ Field-aware Factorization Machine

☐ Result





# Training Dataset

- adult.data (<https://archive.ics.uci.edu/ml/datasets/adult>)
- Prediction task is to determine whether a person makes over **50K** a year
- 15 columns in total
  - *(age, workclass, final weight, education, education number, marital status, occupation, relationship, race, sex, capital gain, capital loss, hours per week, native country)*



# Outline

☒ Introduction

☒ Data

 ☐ **Preprocessing Codes**

☐ Models

☐ Logistic Regression

☐ Degree-2 Polynomial Mapping

☐ Factorization Machine

☐ Field-aware Factorization Machine

☐ Result



# Import libraries

- To begin with, make sure that these libraries are already installed on your computer.
- If not, install these using 'pip install'
  - pandas
  - Pytorch (torch)
  - numpy

```
import pandas as pd
import numpy as np
import torch
from torch import nn
```



# Loading the Dataset

- Load **adult.data**
- Print first five rows to check the content.

```
df = pd.read_csv('data/adult.data', header=None)  
df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K



# Preprocess (1)

- Drop numerical columns
- Encode the data to numerical type

```
from sklearn.preprocessing import LabelEncoder

df_out = df.copy()
df_out.drop(columns=[0, 2, 4, 10, 11, 12], inplace=True)
num_values = []
for col in df_out.columns:
    df_out[col] = LabelEncoder().fit_transform(df_out[col])
    num_values.append(df_out[col].max() + 1)

df_out.head()
```

[9, 16, 7, 15, 6, 5, 2, 42, 2]

	1	3	5	6	7	8	9	13	14
0	7	9	4	1	1	4	1	39	0
1	6	9	2	4	0	4	1	39	0
2	4	11	0	6	1	4	1	39	0
3	4	1	2	6	0	2	1	39	0
4	4	9	2	10	5	2	0	5	0



# Preprocess (2)

- Label encoding
  - LabelEncoder()

Publisher	Advertiser
ESPN	Gucci
CNN	Nike
ABC	Adidas
NBC	Gucci



Publisher	Advertiser
0	0
1	1
2	2
3	0

Number of values in a column = max number in the column + 1



# Preprocess (3)

- Split the dataset into train and test set
- Print their shapes to check the result

```
data_x = df_out.iloc[:, :-1].values
data_y = df_out.iloc[:, -1].values

mask = np.random.rand(df.shape[0]) < 0.9
trn_x = data_x[mask]
trn_y = data_y[mask]
test_x = data_x[~mask]
test_y = data_y[~mask]

print(trn_x.shape, test_x.shape)
print(trn_y.shape, test_y.shape)
```

(29276, 8) (3285, 8)  
(29276,) (3285,)



# Preprocess (4)

- Set batch size
- Create a Dataset class for each dataset
- Create a Dataloader class to utilize the dataset

```
from torch.utils.data import DataLoader, TensorDataset

batch_size = 128
trn_data = TensorDataset(torch.from_numpy(trn_x), torch.from_numpy(trn_y))
trn_loader = DataLoader(trn_data, batch_size, shuffle=True)
test_data = TensorDataset(torch.from_numpy(test_x), torch.from_numpy(test_y))
test_loader = DataLoader(test_data, batch_size)
```





# Outline

☒ Introduction

☒ Data

☒ Preprocessing Codes

 ☐ **Models**

☐ Linear Model

☐ Degree-2 Polynomial Mapping

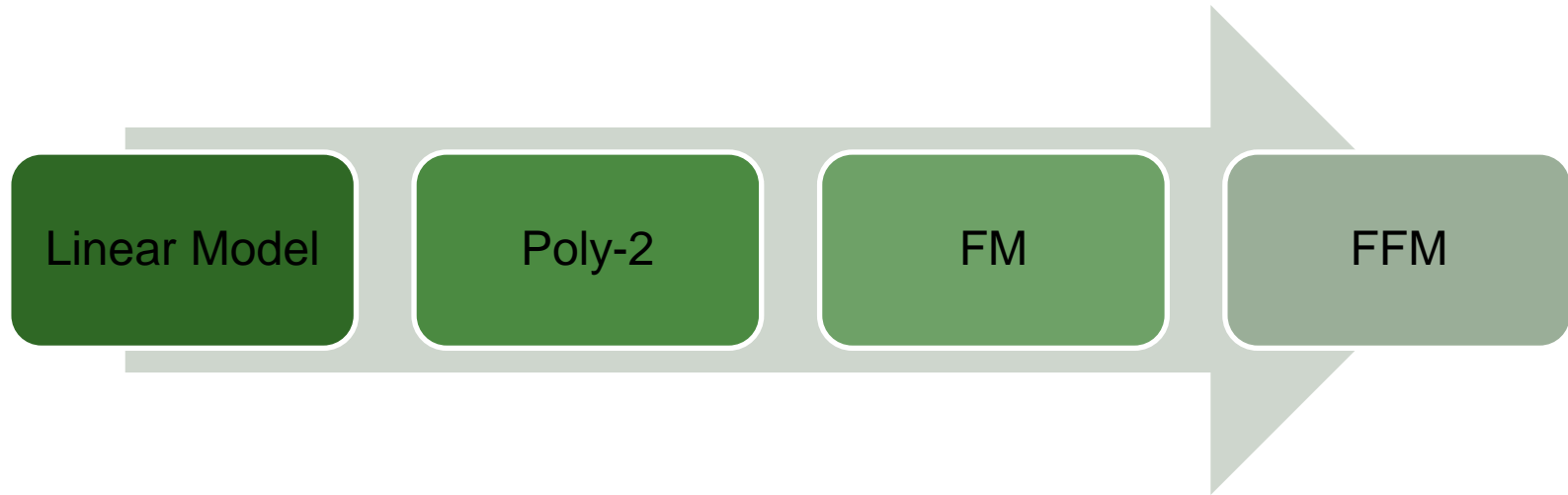
☐ Factorization Machine

☐ Field-aware Factorization Machine

☐ Result




# The Evolution of Models



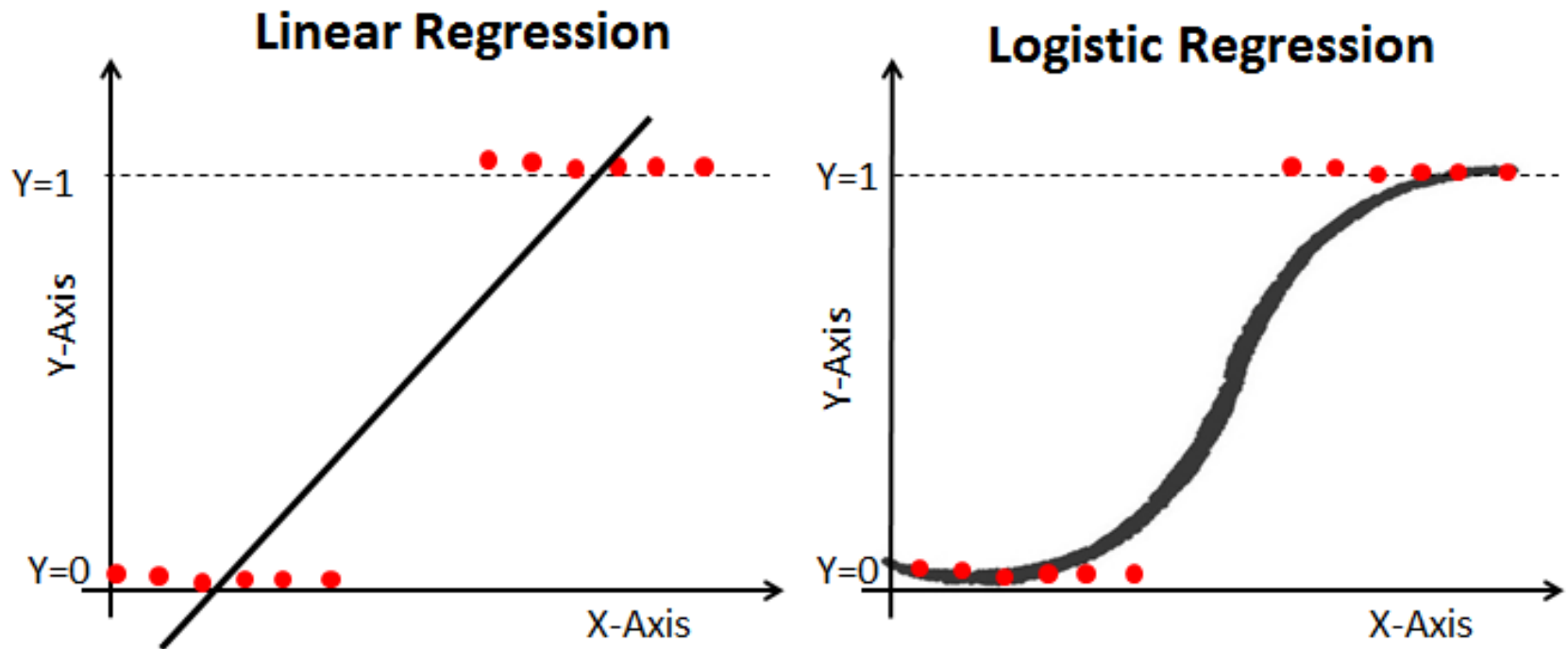


# Outline

- ☒ Introduction
- ☒ Data
- ☒ Preprocessing Codes
- ☒ **Models**
  -  ☐ Linear Model
  - ☐ Degree-2 Polynomial Mapping
  - ☐ Factorization Machine
  - ☐ Field-aware Factorization Machine
- ☐ Result



# Review : Linear Model (LM)





# Review : Logistic Regression (2)

$$P(y = 1|x) = \frac{1}{1 + e^{-\phi(w,x)}}$$

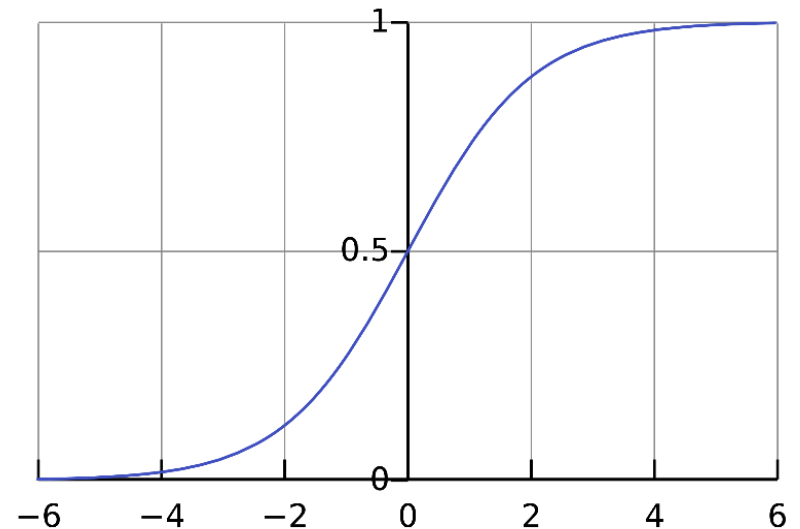
where

$x$  is the feature

$y$  is the label

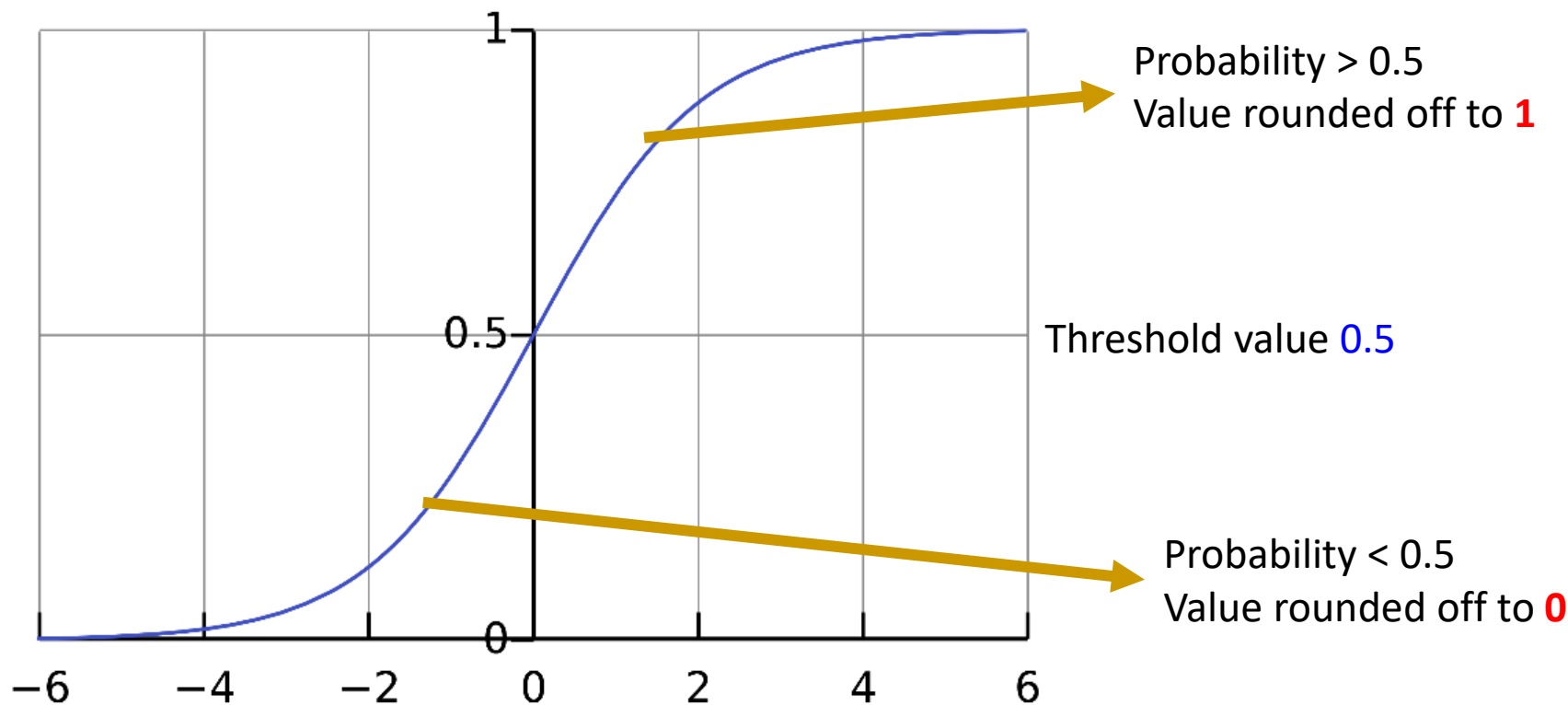
$w$  is a parameter of the model

$\phi$  is a logit





# Review : Logistic Regression (3)





# Review : Logistic Regression (4)

Publisher (P)	Advertiser (A)	Gender (G)
ESPN	NIKE	Male

- $\phi(w, x) = \sum_{j=1}^n w_j^T x_j$

$$\begin{array}{ccccc} \text{●} & + & \text{●} & + & \text{●} \\ w_{ESPN} & & w_{Nike} & & w_{Male} \end{array}$$

We use one-hot representations for features



# Quiz: Logistic Regression

Click (Y)	Publisher(X1)	Advertiser(X2)	Gender(X3)
Yes	ESPN	NIKE	Male
No	NBC	Gucci	Female
No	ESPN	Gucci	Male

- How many parameters the model should learn based on the training set above?





# Quiz: Logistic Regression

Click (Y)	Publisher(X1)	Advertiser(X2)	Gender(X3)
Yes	ESPN	NIKE	Male
No	NBC	Gucci	Female
No	ESPN	Gucci	Male

- How many parameters the model should learn based on the training set above?

☐ 6

☐ ESPN, NBC, NIKE, Gucci, Male, Female



# Logistic Regression Model

## ■ Define linear model

```
class LinearModel(nn.Module):
    def __init__(self, num_values):
        super().__init__()
        weights = []
        for n in num_values:
            weights.append(nn.Embedding(n, 1))
        self.weights = nn.ModuleList(weights)

    def forward(self, x):
        w_out = []
        for i in range(x.size(1)):
            w_out.append(self.weights[i](x[:, i]))
        return torch.sigmoid(torch.cat(w_out, dim=1).sum(dim=1))
```



# Limitation of Logistic Regression

- What is the disadvantage of the model?




# Limitation of Logistic Regression

- What is the disadvantage of the model?
  - The model cannot learn the effect of a feature conjunction
  - If the click through rate of the ads from NIKE on ESPN is particularly high, **it cannot learn the effect well**
    - It only learns either NIKE is very popular for everyone, or ESPN is an effective publisher for all advertisers!



# Outline

- ☒ Introduction
- ☒ Data
- ☒ Preprocessing Codes
- ☒ Models
  - ☒ Linear Model
  -  ☐ **Degree-2 Polynomial Mapping**
  - ☐ Factorization Machine
  - ☐ Field-aware Factorization Machine
- ☐ Result



# Review : Poly-2 Mapping

Publisher (P)	Advertiser (A)	Gender (G)
ESPN	NIKE	Male

- $\phi_{Poly-2}(w, x) = \sum_{j_1, j_2 \in \mathcal{C}_2} w_{j_1, j_2} x_{j_1} x_{j_2}$

$$\begin{array}{ccccc} \text{●} & + & \text{●} & + & \text{●} \\ w_{ESPN, Nike} & & w_{Nike, Male} & & w_{Male, ESPN} \end{array}$$

We use one-hot representations for features



# Quiz: Poly-2

Click (Y)	Publisher(X1)	Advertiser(X2)	Gender(X3)
Yes	ESPN	NIKE	Male
No	NBC	Gucci	Female
No	ESPN	Gucci	Male

- How many parameters the model should learn based on the training set above?



# Quiz: Poly-2

Click (Y)	Publisher(X1)	Advertiser(X2)	Gender(X3)
Yes	ESPN	NIKE	Male
No	NBC	Gucci	Female
No	ESPN	Gucci	Male

- How many parameters the model should learn based on the training set above?

□ 8

- ESPN-NIKE, ESPN-Male, NIKE-Male, NBC-Gucci, NBC-Female, Gucci-Female, ESPN-Gucci, Gucci-Male





# Polynomial-2 Model

## ■ Define poly2 model

```
class Poly2Model(nn.Module):
    def __init__(self, num_values):
        super().__init__()
        weights = []
        for (i, n1) in enumerate(num_values):
            w = []
            for (j, n2) in enumerate(num_values):
                if i < j:
                    w.append(nn.Embedding(n1 * n2, 1))
                else:
                    w.append(nn.Sequential())
            weights.append(nn.ModuleList(w))
        self.weights = nn.ModuleList(weights)
        self.num_values = num_values

    def forward(self, x):
        w_out = []
        for i in range(x.size(1)):
            for j in range(x.size(1)):
                if i < j:
                    w = self.weights[i][j](x[:, i] * self.num_values[j] + x[:, j])
                    w_out.append(w)
        return torch.sigmoid(torch.cat(w_out, dim=1).sum(dim=1))
```



# Limitation of Poly-2

- What is the disadvantage of the model?




# Limitation of Poly-2

- What is the disadvantage of the model?
  - ❑ It considers all feature pairs shown on the dataset.
  - ❑ As a result, **the memory cost** and **computing time** are too heavy and long.
  - ❑ For most datasets, poly-2 model is not tractable due to its heavy costs



# Outline

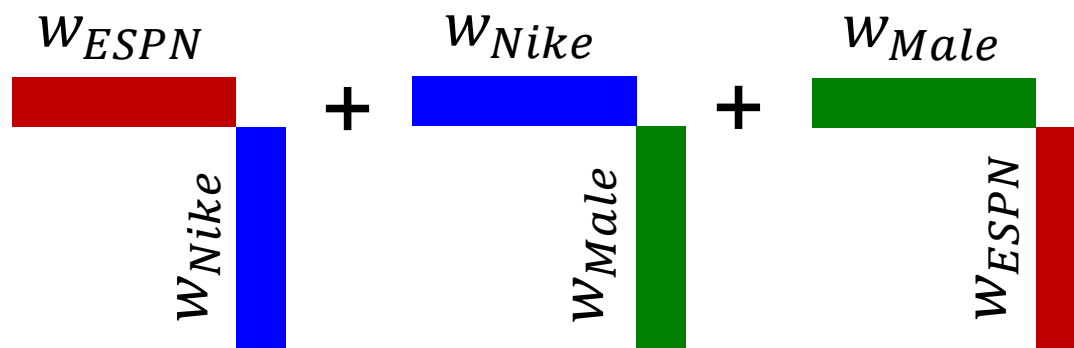
- ☒ Introduction
- ☒ Data
- ☒ Preprocessing Codes
- ☒ Models
  - ☒ Linear Model
  - ☒ Degree-2 Polynomial Mapping
  -  ☐ **Factorization Machine**
  - ☐ Field-aware Factorization Machine
- ☐ Result



# Review : Factorization Machine

Clicked	Publisher (P)	Advertiser (A)	Gender (G)
Yes	ESPN	Nike	Male

- $\phi_{FM}(w, x) = \sum_{j_1, j_2 \in C_2} \langle w_{j_1}, w_{j_2} \rangle x_{j_1} x_{j_2}$



Not scalars, but vectors!



# Factorization Machine Model


## ■ Define FM model

```
class FM(nn.Module):
    def __init__(self, num_values, dim):
        super().__init__()
        weights = []
        for n in num_values:
            weights.append(nn.Embedding(n, dim))
        self.weights = nn.ModuleList(weights)
        self.dim = dim

    def forward(self, x):
        w_out = []
        for i in range(x.size(1)):
            for j in range(x.size(1)):
                if i < j:
                    w1 = self.weights[i](x[:, i]).view(-1, 1, self.dim)
                    w2 = self.weights[j](x[:, j]).view(-1, self.dim, 1)
                    w = torch.bmm(w1, w2).view(-1, 1)
                    w_out.append(w)
        return torch.sigmoid(torch.cat(w_out, dim=1).sum(dim=1))
```



# Outline

- ☒ Introduction
- ☒ Data
- ☒ Preprocessing Codes
- ☒ Models
  - ☒ Linear Model
  - ☒ Degree-2 Polynomial Mapping
  - ☒ Factorization Machine
  -  ☐ **Field-aware Factorization Machine**
- ☐ Result

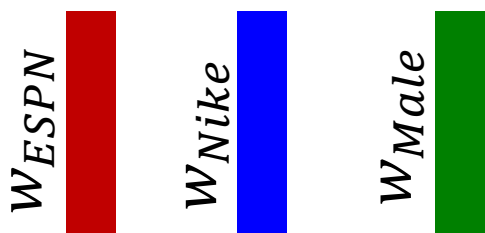


# Idea of Field-aware Factorization Machine (1)

- The model creates vectors for not feature-feature pairs, but feature-field pairs.

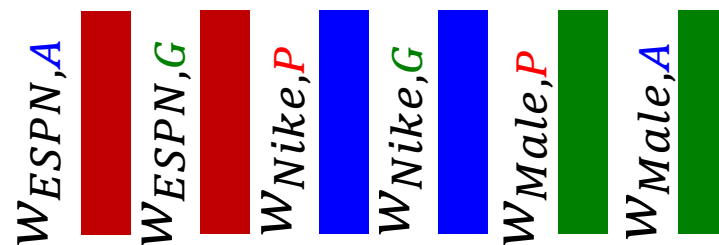
Clicked	Publisher	Advertiser	Gender
Yes	ESPN	Nike	Male

**FM**



combination

**FFM**



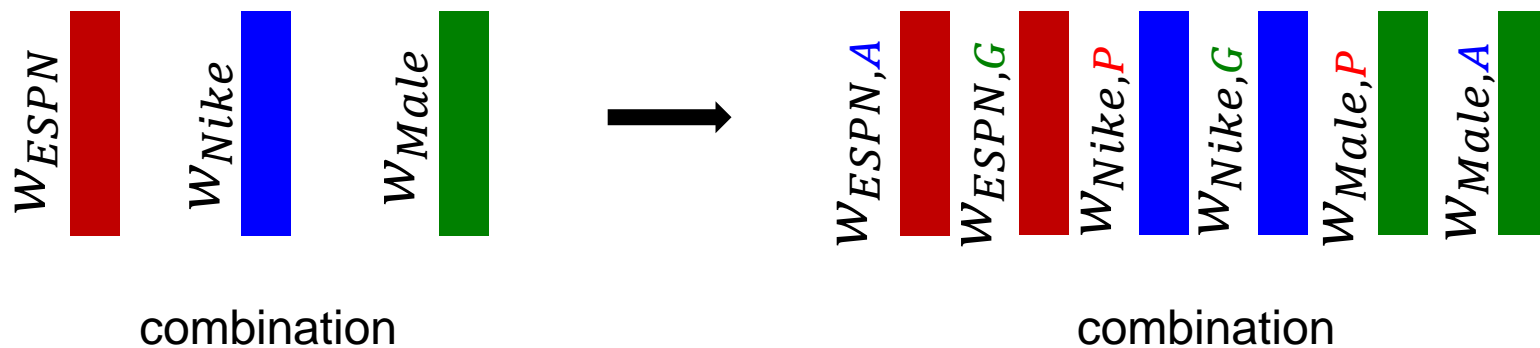
combination





# Idea of Field-aware Factorization Machine (2)

- In FM model, the vectors used when calculating ESPN-Nike pair and ESPN-Male pair are the same.
- For a given item, FFM model uses different feature vectors considering the field of its peer

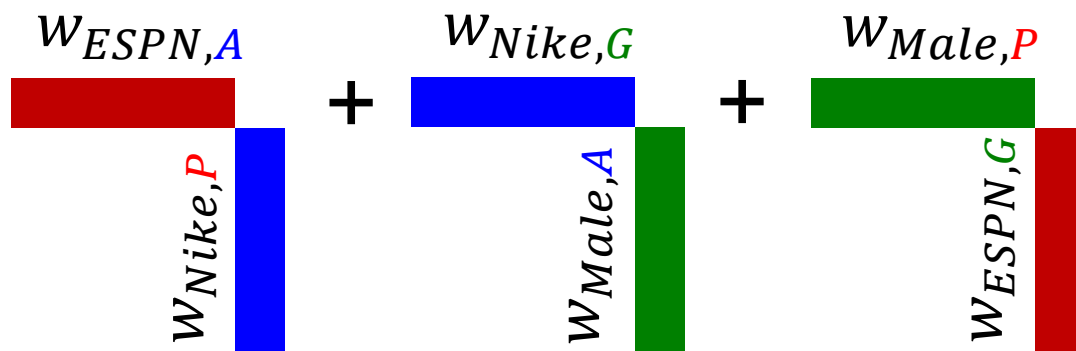




# Review : Field-aware Factorization Machine (FFM)

Clicked	Publisher (P)	Advertiser (A)	Gender (G)
Yes	ESPN	Nike	Male

■  $\phi_{FFM}(w, x) = \sum_{j_1, j_2 \in C_2} \langle w_{j_1, f_2}, w_{j_2, f_1} \rangle x_{j_1} x_{j_2}$





# Field-aware Factorization Machine Model

## ■ Define FFM model

```
class FFM(nn.Module):
    def __init__(self, num_values, dim):
        super().__init__()
        weights = []
        for n in num_values:
            w = []
            for _ in range(len(num_values)):
                w.append(nn.Embedding(n, dim))
            weights.append(nn.ModuleList(w))
        self.weights = nn.ModuleList(weights)
        self.dim = dim

    def forward(self, x):
        w_out = []
        for i in range(x.size(1)):
            for j in range(x.size(1)):
                if i < j:
                    {
                        w1 = self.weights[i][j](x[:, i]).view(-1, 1, self.dim)
                        w2 = self.weights[j][i](x[:, j]).view(-1, self.dim, 1)
                        w = torch.bmm(w1, w2).view(-1, 1)
                    }
                w_out.append(w)
        return torch.sigmoid(torch.cat(w_out, dim=1).sum(dim=1))
```



# Outline

- ☒ Introduction
- ☒ Data
- ☒ Preprocessing Codes
- ☒ Models
  - ☒ Linear Model
  - ☒ Degree-2 Polynomial Mapping
  - ☒ Factorization Machine
  - ☒ Field-aware Factorization Machine

 ☐ **Result**



# Training Process (1)

- Implement a function which transforms predicted probability to 0 or 1 label

```
def to_corrects(preds, labels):  
    return ((preds - 0.5).ceil() == labels).float().numpy()
```



# Training Process (2)

- Set epochs, dimension, and model to use
- Call loss function and optimizer

```
from torch import optim

epochs = 100
dim = 8

model = LinearModel(num_values)
# model = Poly2Model(num_values)
# model = FM(num_values, dim)
# model = FFM(num_values, dim)

loss_func = nn.BCELoss()
optimizer = optim.Adam(model.parameters())
```



# Training Process (3)

## ■ Train the model with multiple epochs

```
for e in range(epochs):
    model.train()
    losses, corrects = [], []
    for x, y in trn_loader:
        y_pred = model(x)
        loss = loss_func(y_pred, y.float())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
        corrects.append(to_corrects(y_pred, y))
    trn_loss = np.mean(losses)
    trn_acc = np.mean(np.concatenate(corrects))

    model.eval()
    losses, corrects = [], []
    for x, y in test_loader:
        y_pred = model(x)
        loss = loss_func(y_pred, y.float())
        losses.append(loss.item())
        corrects.append(to_corrects(y_pred, y))
    test_loss = np.mean(losses)
    test_acc = np.mean(np.concatenate(corrects))
```



# Result (1)

- For each model, run the code and check the output
  - The losses decrease while accuracies increase

0	0.6328	0.6748	0.5815	0.7035	Epoch #
1	0.5339	0.7381	0.5134	0.7492	
2	0.4845	0.7633	0.4752	0.7720	
3	0.4547	0.7801	0.4505	0.7854	
4	0.4347	0.7926	0.4323	0.8006	Training Loss
5	0.4198	0.8040	0.4190	0.8094	
6	0.4089	0.8079	0.4086	0.8125	
7	0.4000	0.8116	0.4005	0.8155	
8	0.3932	0.8169	0.3940	0.8213	
9	0.3876	0.8204	0.3887	0.8225	Training Accuracy
10	0.3830	0.8232	0.3844	0.8244	
11	0.3791	0.8250	0.3809	0.8253	
12	0.3760	0.8255	0.3780	0.8271	
13	0.3735	0.8256	0.3756	0.8280	Test Loss
14	0.3714	0.8263	0.3736	0.8301	
15	0.3693	0.8271	0.3718	0.8314	
16	0.3677	0.8275	0.3705	0.8314	
17	0.3666	0.8275	0.3693	0.8307	Test Accuracy
18	0.3654	0.8278	0.3683	0.8314	
19	0.3645	0.8287	0.3675	0.8317	
20	0.3637	0.8290	0.3667	0.8332	





## Result (2)

- We compare the results of the four models
  - Test loss is smaller in Linear and FM
  - Training loss is smaller in Poly2 and FFM
- Thus, it is not always better to use a complex model if the task is simple

Model	TrnLoss	TrnAcc	TestLoss	TestAcc
Linear	0.357	0.833	0.366	0.825
Poly2	0.329	0.845	0.402	0.823
FM	0.347	0.838	0.376	0.826
FFM	0.335	0.842	0.408	0.821



# What You Need To Know

- How to predict a label from categorical datasets
- The difference between training models and how one overcomes another's limitation
- How to implement them in pytorch



# Questions?