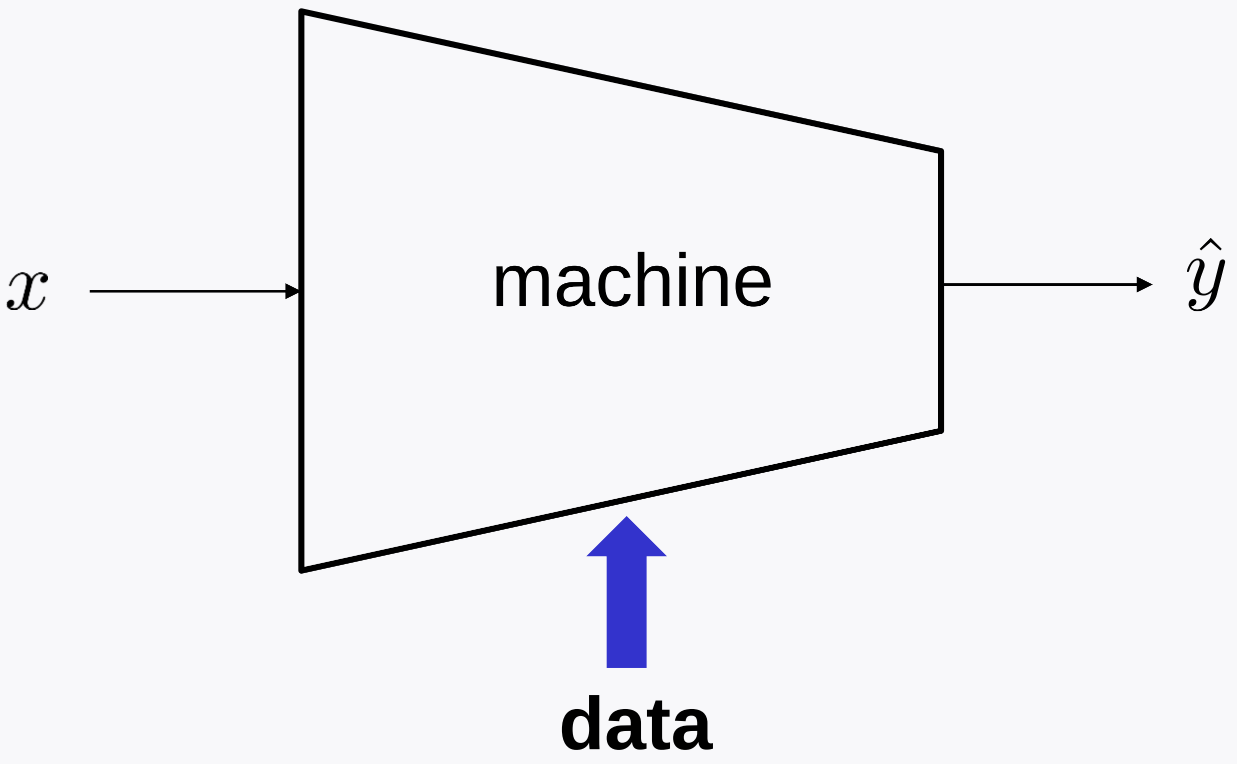
Advanced techniques

Practice Session 4

Changho Suh

January 23, 2024

Recap: Machine learning



Recap: Training via optimization

Optimization problem:

$$\min_{w} \sum_{i=1}^{m} \ell(y^{(i)}, f_w(x^{(i)}))$$

Two things to choose:

1. loss function $\ell(\cdot) \longrightarrow$ squared error loss, cross entropy loss
2. model $f_w(\cdot) \longrightarrow$ Perceptron, DNN

Recap: How to solve?

$$\min_{w} \sum_{i=1}^{m} \ell(y^{(i)}, f_w(x^{(i)}))$$

Gradient descent:

$$w^{(t+1)} \leftarrow w^{(t)} - \alpha \boxed{\nabla J(w^{(t)})}$$

backpropagation

A prominent variant of gradient descent:

Adam

Recap: Data organization

m is a deciding factor for the **ratio** of data split:

4 regimes: Small, middle, large and ultra-large

val set dist. ~ test set dist. ~ target dist.

Recap: Generalization techniques

$$\min_w \sum_{i=1}^{m} \ell(y^{(i)}, f_w(x^{(i)})) \;\; +\lambda\|w\|^2$$

$$\boxed{\text{GD:} \quad w^{(t+1)} \leftarrow w^{(t)} - \alpha \nabla J(w^{(t)})}$$

Generalization techniques

1. Regularization
2. Data augmentation
3. Early stopping
4. Dropout

Recap: Weight initialization

$$\min_{w} \sum_{i=1}^{m} \ell(y^{(i)}, f_w(x^{(i)}))$$

GD: $w^{(t+1)} \leftarrow w^{(t)} - \alpha \nabla J(w^{(t)})$

Question: How to choose $w^{(0)}$?

1. Xavier's initialization

2. He's initialization (under ReLU)

Recap: Techniques for training stability

$$\min_{w} \sum_{i=1}^{m} \ell(y^{(i)}, f_w(x^{(i)}))$$

GD: $w^{(t+1)} \leftarrow w^{(t)} - \alpha \nabla J(w^{(t)})$

1. Learning rate decaying
2. Batch normalization

Recap: Hyperparameter search

$$\min_{w} \sum_{i=1}^{m} \ell(y^{(i)}, f_w(x^{(i)}))$$

GD: $w^{(t+1)} \leftarrow w^{(t)} - \alpha \nabla J(w^{(t)})$

L of layers, # $n^{[\ell]}$ of hidden neurons, activation
learning rate, betas, batch size, # *T* of epochs,
regularization factor, dropout rate, …

Recap: Cross validation

4 fold cross validation:

val	train	train	train	test	val_1
train	val	train	train	test	val_2
train	train	val	train	test	val_3
train	train	train	val	test	val_4

Consider the average:

$$\mathsf{val\ loss} = \frac{\mathsf{val}_1 + \mathsf{val}_2 + \mathsf{val}_3 + \mathsf{val}_4}{4}$$

Choose a hyperparameter that minimizes the **average loss.**

Outline

Will do coding exercises for all the techniques:

- Data organization (train/validation/test sets)
- Generalization techniques
- Weight initialization
- Techniques for training stability
- Hyperparameter search
- Cross validation

Will do this in the context of a simple task:

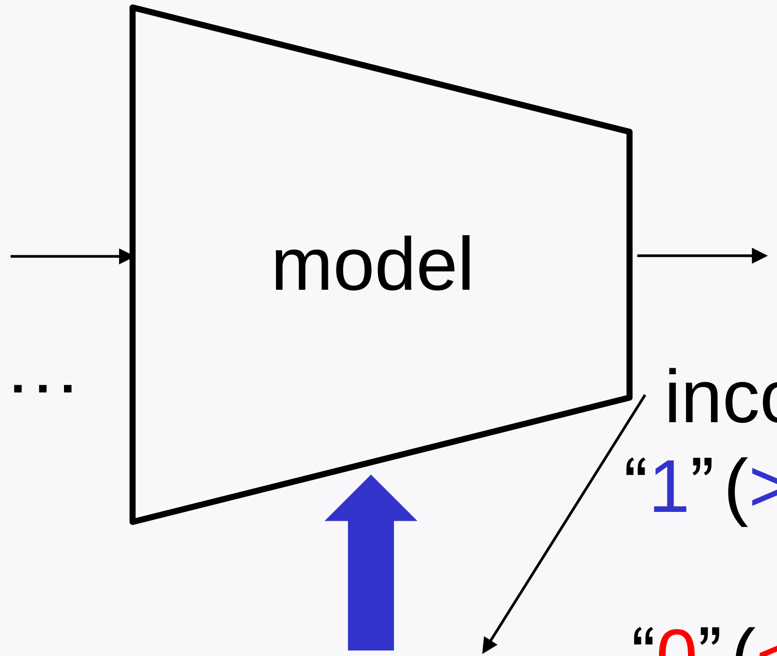
Adult income classification

Adult income classification

x

age, education, …

(12 attributes)



$\hat{y}$

income:

"1" (> $50K)

or

"0" (< $50K)

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$$

Data loading

```
pip install shap
```

```
from shap.datasets import adult
X, y = adult()
```

```
print(X.shape)
print(y.shape)
```

```
(32561, 12)
(32561,)
```

```
print(type(X))
print(type(y))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'numpy.ndarray'>
```

Data loading

```
pip install shap
from shap.datasets import adult
X, y = adult()
print(X)
print(y)
```

numerical data

Remaining is categorical

	Age	Workclass	Education-Num	Marital Status	Occupation
0	39.0	7	13.0	4	1
1	50.0	6	13.0	2	4
2	38.0	4	9.0	0	6
3	53.0	4	7.0	2	6
4	28.0	4	13.0	2	10
...	...	...	...	...	...
32556	27.0	4	12.0	2	13
32557	40.0	4	9.0	2	7
32558	58.0	4	9.0	6	1
32559	22.0	4	9.0	4	1
32560	52.0	5	9.0	2	4

Relationship	Race	Sex	Capital Gain	Capital Loss	Hours per week	Country
0	4	1	2174.0	0.0	40.0	39
4	4	1	0.0	0.0	13.0	39
0	4	1	0.0	0.0	40.0	39
4	2	1	0.0	0.0	40.0	39
5	2	0	0.0	0.0	40.0	5
...	...	...	...	...	...	...
5	4	0	0.0	0.0	38.0	39
4	4	1	0.0	0.0	40.0	39
1	4	0	0.0	0.0	40.0	39
3	4	1	0.0	0.0	20.0	39
5	4	0	15024.0	0.0	40.0	39

[32561 rows x 12 columns]

```
[False False False ... False False  True]
```

Preprocessing

```
X, y = adult()
```

```
numerical_columns = ['Age','Education-Num','Capital Gain','Capital Loss','Hours per week']
categorical_columns = ['Workclass','Marital Status','Occupation','Relationship','Race','Sex','Country']
```

```
from sklearn.preprocessing import StandardScaler # for normalization
```

```
# Normalization of numerical data
for column in numerical_columns:
    scaler = StandardScaler()
    X[column] = scaler.fit_transform(X[column].values.reshape(-1,1))
```

```
# Data type change of categorical data
for column in categorical_columns:
    X[column] = X[column].astype('category')
```

```
import pandas as pd # for one-hot encoding
```

```
# One-hot encoding of categorical data
X = pd.get_dummies(X)
```

Preprocessing: data frame → numpy

```
# Conversion of data frame to numpy
X = X.values
```

```
# Converision: {False, True} --> {0., 1.}
y = y.astype(float)
```

```
print(X.shape)
print(y.shape)
print(y)
```

```
(32561, 50)
(32561,)
[0. 0. 0. ... 0. 0. 1.]
```

Data organization: train-val-test split

Suppose we want:

train:val:test = 8:1:1

to ensure the same distribution

```
from sklearn.model_selection import train_test_split
X_,X_test,y_,y_test = train_test_split(X,y,test_size=1/10,stratify=y)
X_train,X_val,y_train,y_val = train_test_split(X_,y_,test_size=1/9,stratify=y_)
```

```
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
(26048, 50)
(3256, 50)
(3257, 50)
```

```
print(sum(y_train)/y_train.shape)
print(sum(y_val)/y_val.shape)
print(sum(y_test)/y_test.shape)
```

```
[0.24082463]
[0.24078624]
[0.24071231]
```

Start with the simplest model

Logistic regression:

```
from sklearn.linear_model import LogisticRegression

model_LR = LogisticRegression()

# training
model_LR.fit(X_train, y_train)

# evaulation
val_acc = model_LR.score(X_val, y_val)

print(val_acc)
```

```
0.85104422604422
```

Saving a sklearn model

```
from joblib import dump

dump(model_LR, 'LR_sample.joblib')
```

LR_sample.joblib seconds ago 1.15 kB

Loading a saved model

```
from joblib import dump

dump(model_LR, 'LR_sample.joblib')
```

```
from joblib import load

loaded_model_LR = load('LR_sample.joblib')
```

Look ahead

Will apply many of the advanced techniques to improve performance.