Case Study: Regression

Machine Learning

Course Outline

Assignments are submitted at the same date as the next assignment comes out

Wk	Lecture 1	Lecture 2	Individual Assignment	Project Assignment
1	Case Study: Regression	Case Study: Classification	A1: Car Price Prediction	
2	Batch Gradient Descent	Stochastic / Mini-Batch Gradient Descent		
3	Regularization	Binary Logistic Regression	A2: Car Price Prediction II	
4	Multinomial Logistic Regression	Gaussian Naive Bayes		
5	Multinomial Naive Bayes	K-Nearest Neighbors / Support Vector Machine	A3: Car Price Prediction III	
6	Decision Tree	Bagging / Random Forests / Ada Boosting / Gradient Boosting		
7	**No class**	**Midterm Exam**		Phase 1: Reading paper round 1 (KDD)
8	K-Means Clustering	Gaussian Mixture		Phase 1: Reading paper round 2 (KDD)
9	Principal Component Analysis	PyTorch Linear Regression		Phase 2: Proposal - Paper writing (Intro, Related Work, Method)
10	PyTorch Logistic Regression	**Project Proposal Presentation**		
11	Convolutional Neural Network	Recurrent Neural Network		Phase 3: Experiment - Paper writing (Intro, Related Work, Method, Results)
12	Reinforcement Learning	Project Time		
13	**Project Time**	**Project Progress Presentation**		
14	**Project Time**	**Project Time**		Phase 4: Conclusion - Paper writing (Abstract, Intro, Related Work, Method, Results, Discussion, Conclusion)
15	**No class**	**Final Exam**		
16	**No class**	**Final Project Presentation**		

Background - Regression

- Predict **blood pressure level** using **BMI?**
- Predict **GDP** using **egg price, gold price, oil price?**
- Predict **life expectancy** using **population, GDP?**

We called population, GDP as **features** (a.k.a predictors, independent variables), and life expectancy as **labels** (a.k.a targets, dependent variables)

Notice all the **labels** here are **continuous** values?

Regression is a **supervised** algorithm to *predict* **continuous** values

- Labels must be continuous; features can be categorical or continuous
- Both features and labels can be univariate or multivariate
- *Supervised* - has both features and labels; *Unsupervised* - only has features
- *Continuous* - opposite of categorical variables, e.g., gold price is continuous while gender is categorical
    - Continuous value can be change to categorical variables by binning!

Big picture of ML

Remember: Splitting > Imputation > Scaling

Step 2 to 5 - can be done iteratively and order may swap, it's ok

1. Load data ->

- CSV, JSON, Database
- Renaming
- Label encoding
- Train / dev / test set

2. Exploratory Data Analysis ->

- Check class imbalance
- Countplot
- Distribution plot
- Boxplot
- Scatter plot
- Correlation Matrix
- Predictive Power Score

3. Feature engineering ->

- Dimensionality reduction
- Feature splitting (e.g., date)
- Creating features (e.g., some equation)

4. Feature selection ->

- Select your X (features) and y (target)
- In ML, it's better to choose X
- In DL, we usually just input all features

5. Preprocessing

- Null values
- Outliers
- Fix class imbalance (up-/down-sampling
- Typos / Entry errors / Duplicates / IDs
- Scaling (min-max; standardize)

6. Model selection ->

Supervised
- Regression
- Classification

Unsupervised
- Clustering
- Dimensionality reduction

Reinforcement
- PPO
- Q-learning

- **Cross-validation**
  - With/Without Sampling
- **Grid search**

7. Testing ->

Apply your best model on your test set

METRIC
- **Regression** (r^2, MSE)
- **Classification** (recall, precision, f1)
- **Clustering** (inertia)
- **Dimensionality reduction** (mean squared distance between the original data and the reconstructed data)
- **Reinforcement learning** (cumulative rewards)

- Overfitting
- Underfitting

- MLFlow
- Wandb
- Tensorboard

8. Analysis ->

Analyze your model, e.g., **feature importance**

9. Inference ->

Apply your best model on some unseen data, and see whether it makes sense

10. Deployment

- HTML/CSS/JS

- Flask
- Django
- FastAPI

- Docker

TOOLS
- **Python**, R - programming tool
  - **NumPy** (matrix manipulation), **Pandas** (Excel-like), **Matplotlib/Seaborn** (visualization), **Sklearn** (machine learning), **PyTorch** (deep learning), **Huggingface** (models)
- **Tableau**, **Power BI** - Business Intelligence (BI) tools
- **Microsoft Azure, Rapidminer, Weka** - data science and machine learning tools
- **SPSS**, **SAS, JASP** - statistical tool

TOP VENUES
1. *ML (KDD)*
2. *DL (ICML, NIPS, ICMR)*
3. *NLP (ACL, EMNLP)*
4. *CV (CVPR, ICCV)*

1. Load data

- Getting data
  - www.kaggle.com/datasets
  - API, e.g., Instagram API, Twitter API, Weather API, Stock Price API
  - Google Dataset search
  - Collect data using sensors
- Data format: CSV, JSON, Databases (SQL)

```
df = pd.read_csv('data/Life_Expectancy_Data.csv')
```

Figure: Python can easily read your csv file

2. Exploratory Data Analysis

- Perhaps the most important step - understand your data
- **Label encoding**: encode so we can visualize properly, as well as performing correlation etc.
  - categorical variables into integers, e.g., male => 0; female => 1
- Univariate
  - Countplot
  - Distribution plot
- Multivariate
  - Boxplot
  - Scatter plot
  - Correlation Matrix
  - Predictive Power Score

One-hot encoding

- Recall label encoding which we turn categories into 0, 1, 2 etc.
- When we have **more than two categories**, if we encode into 0, 1, 2
  - we create a unintentional order, i.e., the model "may" think that $0 < 1 < 2$
- Possible solution: **one hot encoding**
  - E.g., Male, Female, Unknown ⇒ [1, 0, 0] if male; [0, 1, 0] if female
  - Limitation
    - what if we have like 5000 categories....
    - one hot encode this will result in 5000 columns --> too much! -> Two choices:
      - Group these categories into bigger categories, and then one-hot encode
      - Do label encoding anyway......but note the possible order effect
- **Tips**: one thing you need to know is that you can always cut down one column
  - [1, 0, 0], [0, 1, 0], [0, 0, 1] is same as [1, 0], [0, 1], [0, 0] by setting `'drop_first=True'`

3. Feature engineering

- **Dimensionality reduction** - reduce dimensions, e.g, Principal Component Analysis (maximizes variances), Discriminant Analysis (maximizes separability)
- **Feature splitting** - Jan 26, 2023 ===> Monday or January
- **Creating features (e.g., some equation)** - combining the total sales from each sale department

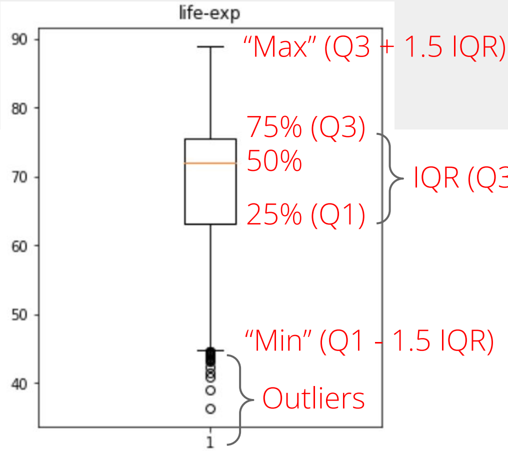
Note: After such engineering, you can always go back to exploratory analysis

4. Feature selection

- Choose the most salient X
  - Rule of thumb: Good features MUST NOT BE correlated, i.e., independent
  - Rule of thumb: Correlation is not causation; don't pick features using correlation only; it should make sense!
    - E.g., *Number of trees is correlated with number of divorce*
  - Rule of thumb: For ML, less features are usually better (but NOT necessarily for DL)
- Specify the y
- Split train / test
  - Rule of thumb: Always split BEFORE preprocessing, to prevent data leakage
    - Can be done in this order: (1) splitting, (2) imputation, (3) scaling

5. Preprocessing

Numbers: 4, 5, 6, 8, **8, 9,** 10, 11, 13, 18
50% = (8 + 9) / 2 = 8.5
75% = 10, **11, 13**, 18 = (11 + 13) /2 = 12
25% = 4, **5, 6**, 8 = (5 + 6) / 2 = 5.5
IQR = Q3 - Q1 = 12 - 5.5 = 6.5
Min = Q1 - 1.5 (IQR) = 5.5 - 1.5 (6.5) = -4.25
Max = Q3 + 1.5(IQR) = 12 + 1.5 (6.5) = 21.25



- **Imputing missing values**
  - For missing "numbers",
    - Replace with average | median | regression results | 0
  - For missing "categories"
    - Replace with mode | ratio | "No category"

      5 M, 6F == 5/6
  - 1st rule of thumb: replacing should not drastically change the distribution
  - 2nd rule of thumb: don't delete if you are not sure what you are dealing with
  - 3rd rule of thumb: replace ONLY features, NOT labels. Don't create fake labels.
- **Outliers**
  - Use visualization such as box plot or z-score
  - Rule of thumb: Don't just delete using box plot rule. Use your common sense to understand what is possible, errors, and impossible
- **Typos / Entry errors / Duplicates / IDs**

5. Preprocessing - scaling

- Scale your features
  - Help the model to learn faster
  - **Standardization**
    - (x - mean) / std
    - when your data follows normal distribution
  - **Normalization**
    - (x - x_min) / (x_max - x_min)
    - when your data DOES NOT follow normal distribution (e.g., audio, signal, image)
- Rule of thumb: Scale after you split and fill all the missing values
- Rule of thumb: Scale your test set using training distribution, NOT testing distribution
- Rule of thumb: DON'T scale your categorical features

6. Model selection - many algorithms

- Compare algorithms - so many algorithms
  - Linear Regression, Random Forest Regressor, etc.
  - Within these algorithms, many parameters

Never use testing data before anything (It will modify the model that can alter it)

- It's not ok to use test-set to compare
  - We use **cross-validation**!

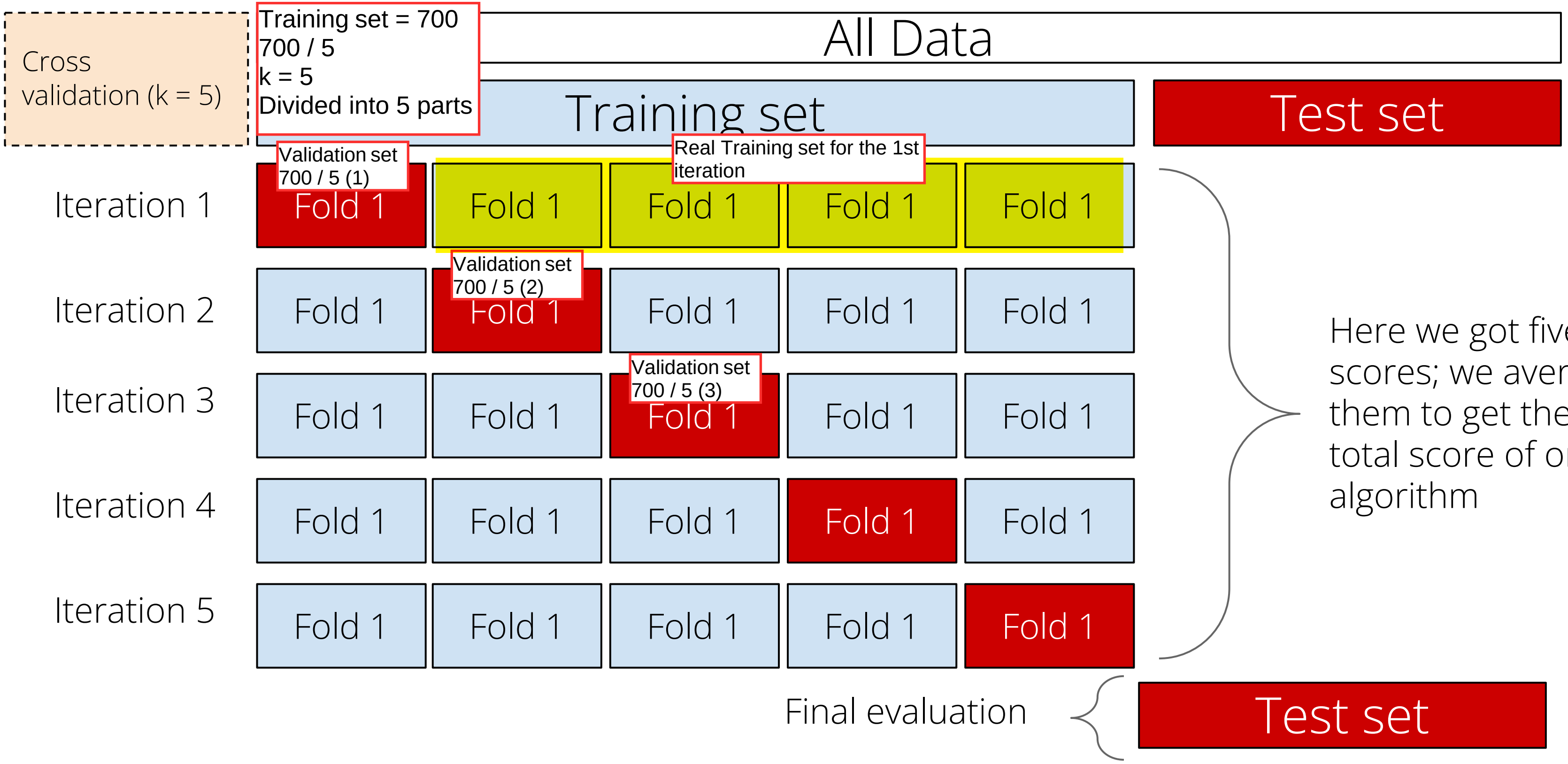
N = 900
Train = 700
Test = 200 (Never touch it until the end)

Use different models to test using train and compare to the Test

How to split?
- Use no repletition method
- Do many times & augment it
  - ite1: A: 2.09 B: 1.08
  - ite2: A: 3.02 B: 2.09
- Then take the mean

K Fold Cross-validation

6. Model selection - cross validation



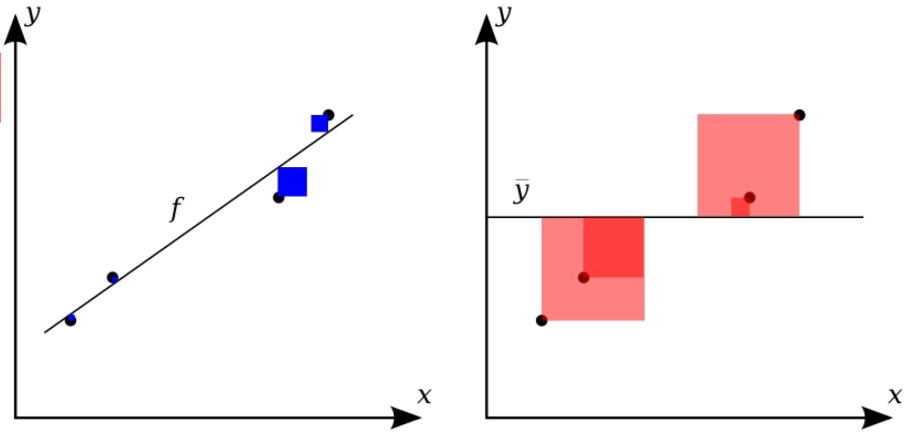
6. Model selection - metrics

Regression

- R^2(R-squared) How reliable of the model compared to the mean
- MSE (Mean Squared Error)
- RMSE (Root Mean Squared Error)

$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \widehat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

R-Squared range from worst to best : (-..., 1)
How it matter?
-... is worst
0 equals to the mean
1 is much better than the mean
- It doesn't mean it's good



$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 \qquad SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2,$$

$\widehat{y}$ The same notation

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

6. Model selection - metrics

y	y^
3	1
-0.5	2
5	5
-2	-3
1	0

MSE =
Square root MSE =
R^2 =

6. Model selection - grid search

- Once we know which algorithm is best, we can further do cross-validation on that only algorithm with different parameters to find the best version

- This is called "**Grid Search**"

learning rate = [0.01, 0.05, 0.1, 0.5]
algorithm = [linear reg, naive bayes]
max_iter = [10, 50, 100, 500]

What combination of these hyperparameters give the most accurate results?
Solution: Use 'Grid Search'

7, 8, 9, 10: Test, Analysis, Inference, Deploy

Test - test your model with the test set

Analysis - analyze your model, e.g., feature importance, where error comes from

Inference - test your model with some unseen data and see whether it makes sense

Deploy - deploy to web using Django, Flask, FastAPI, Streamlit, etc.