Lecture 10 Introduction to Machine Learning and Linear Regression

Overview of the whole picture

Possible hierarchies of machine learning concepts:

- **Problems**: Supervised Learning(Regression,Classification), Unsupervised Learning (Dimension Reduction, Clustering), Reinforcement Learning (Not covered in this course)

- **Models**:
  - (Supervised) Linear Regression, Logistic Regression, K-Nearest Neighbor (kNN) Classification/Regression, Decision Tree, Random Forest, Support Vector Machine, Ensemble Method, Neural Network...
  - (Unsupervised) K-means,Hierachical Clustering, Principle Component Analysis, Manifold Learning (MDS, IsoMap, Diffusion Map, tSNE), Auto Encoder...

- **Algorithms**: Gradient Descent, Stochastic Gradient Descent (SGD), Back Propagation (BP),Expectation–Maximization (EM)...

For the same **problem**, there may exist multiple **models** to discribe it. Given the specific **model**, there might be many different **algorithms** to solve it.

Why there is so much diversity? The following two fundamental principles of machine learning may provide theoretical insights.

Bias-Variance Trade-off (https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229): Simple models -- large bias, low variance. Complex models -- low bias, large variance

No Free Lunch Theorem (https://analyticsindiamag.com/what-are-the-no-free-lunch-theorems-in-data-science/#:~:text=Once%20Upon%20A%20Time,that%20they%20brought%20a%20drink): (in plain language) There is no one model that works best for every problem. (more quantitatively) Any two models are equivalent when their performance averaged across all possible problems. --Even true for optimization algorithms (https://en.wikipedia.org/wiki/No_free_lunch_in_search_and_optimization).

Linear Regression

Recall the basic task of **supervised learning**: given the *training dataset* $(x^{(i)}, y^{(i)}), i = 1, 2, \ldots, N$ with $y^{(i)} \in \mathbb{R}^q$ (for simplicity, assume $q = 1$) denotes the *labels*, the supervised learning aims to find a mapping $y \approx \mathbf{f}(x) : \mathbb{R}^p \to \mathbb{R}$ that we can use it to make predictions on the test dataset.

Model Setup

Model assumption 1: Linear Mapping Assumption.

$$y \approx \mathbf{f}(x) = \beta_0 + \beta_1 x_1 + .. + \beta_p x_p = \tilde{x}\beta,$$

$$\tilde{x} = (1, x_1, .., x_p) \in \mathbb{R}^{1\times(p+1)}, \beta = (\beta_0, \beta_1, .., \beta_p)^T \in \mathbb{R}^{(p+1)\times 1}.$$

Here β is called regression coefficients, and β_0 specially referred to intercept.

Using the whole training dataset, we can write as

$$Y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(N)} \end{pmatrix} \approx \begin{pmatrix} \mathbf{f}(x^{(1)}) \\ \mathbf{f}(x^{(2)}) \\ \cdots \\ \mathbf{f}(x^{(N)}) \end{pmatrix} = \begin{pmatrix} \tilde{x}^{(1)}\beta \\ \tilde{x}^{(2)}\beta \\ \cdots \\ \tilde{x}^{(N)}\beta \end{pmatrix} = \begin{pmatrix} \tilde{x}^{(1)} \\ \tilde{x}^{(2)} \\ \cdots \\ \tilde{x}^{(N)} \end{pmatrix} \beta = \tilde{X}\beta,$$

where

$$\tilde{X} = \begin{pmatrix} 1 & x_1^{(1)} & \cdots & x_p^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_p^{(2)} \\ \cdots & & & \\ 1 & x_1^{(N)} & \cdots & x_p^{(N)} \end{pmatrix}$$

is also called the augmented data matrix.

Model assumption 2: Gaussian Residual Assumption (L^2 loss assumption)

$$y^{(i)} = \tilde{x}^{(i)}\beta + \epsilon^{(i)}, i = 1, 2, .., N$$

The residuals or errors $\epsilon^{(i)}$ are **assumed** as independent Gaussian random variables with identical distribution $\mathcal{N}(0, \sigma^2)$ which has mean 0 and standard deviation σ.

From the density function of Gaussian distribution, the prabability to observe $\epsilon^{(i)}$ within the small interval $[z, z + \Delta z]$ is roughly

$$\frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{z^2}{2\sigma^2})\Delta z.$$

From the data, we know indeed $z = y^{(i)} - \tilde{x}^{(i)}\beta$. Therefore, given $x^{(i)}$ as fixed, the probability density (likelihood) to observe $y^{(i)}$ is roughly

$$l(y^{(i)}|x^{(i)}, \beta) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(y^{(i)} - \tilde{x}^{(i)}\beta)^2}{2\sigma^2}).$$

Using the *independence* assumption, the overall likelihood to observe the response data $y^i (i = 1, 2, \ldots, N)$ is

$$\mathcal{L}(y^{(i)}, 1 \le i \le N|\beta, x^{(i)}) = \prod_{i=1}^{N} l(y^{(i)}|x^{(i)}, \beta)$$

The famous **Maximum Likelihood Estimation** theory in statistics **assumes** that we aim to find the unknown parameter β that maximizes the $\mathcal{L}(\beta; x^{(i)}, y^{(i)}, 1 \le i \le N)$ by treating $x^{(i)}$ and $y^{(i)}$ as fixed numbers.

Equivalently, as the function of β, we can maximize

$$\ln \mathcal{L} = \sum_{i=1}^{N} \ln l(y^{(i)}|\beta, x^{(i)}).$$

By removing the constants, we finally arrives at the **minimization** problem of L^2 loss function

$$L(\beta) = \sum_{i=1}^{N}(y^{(i)} - \tilde{x}^{(i)}\beta)^2 = ||Y - \tilde{X}\beta||_2^2.$$

The optimal parameter

$$\hat{\beta} = \text{argmin} L(\beta)$$

is also called the ordinary least square (OLS) estimator in statistics community.

We also have the prediction

$$\hat{y}^{(i)} = \tilde{x}^{(i)}\hat{\beta}.$$

Algorithm: Normal Equation

To solve the critical points, we have $\nabla L(\beta) = 0$.

$$\frac{\partial L}{\partial \beta_0} = 2 \sum_{i=1}^{N} (\tilde{x}^{(i)} \beta - y^{(i)}) = 0,$$

$$\frac{\partial L}{\partial \beta_k} = 2 \sum_{i=1}^{N} x_k^{(i)} (\tilde{x}^{(i)} \beta - y^{(i)}) = 0, \quad k = 1, 2, .., p.$$

In Matrix form, it can be expressed as (left as exercise)

$$\tilde{X}^T \tilde{X} \beta = \tilde{X}^T Y,$$

also called the **normal equation** of linear regression. Then the OLS estimator can be solved as

$$\hat{\beta} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T Y.$$

Geometrical Interpretation (https://en.wikipedia.org/wiki/Ordinary_least_squares)

Denote $\tilde{X} = (\tilde{X}_0, \tilde{X}_1, .., \tilde{X}_p)$, then $\tilde{X}\beta = \sum_{k=0}^{p} \beta_k \tilde{X}_k$. We require that the residual $Y - \tilde{X}\beta$ is vertical to the plane spanned by $\tilde{X}_k$, which yields

$$\tilde{X}_k^T (Y - \tilde{X}\beta) = 0, \quad k = 0, 1, \ldots, p$$

Extensions: Regularization, Ridge Regression and LASSO

Recall the likelihood function -- we interpret it as the probability of observing the response data, given the parameter β as fixed, i.e. conditional probability

$$\mathcal{P}(y^{(i)}, 1 \leq i \leq N|\beta, x^{(i)}) = \prod_{i=1}^{N} l(y^{(i)}|x^{(i)}, \beta)$$

Now we take a bayesian approach -- assume β is the random variable with **prior distirbution** $\mathcal{P}(\beta)$. Then the **posterior distribution** of β given the data is

$$\mathcal{P}(\beta|x^{(i)}, y^{(i)}, 1 \leq i \leq N) \propto \mathcal{P}(\beta)\mathcal{P}(y^{(i)}, 1 \leq i \leq N|\beta, x^{(i)}).$$

The **Bayesian** estimation aims to maximaze the posterior distribution. Note that

$$\mathrm{argmax}_{\beta}\mathcal{P}(\beta|x^{(i)}, y^{(i)}, 1 \leq i \leq N) = \mathrm{argmax}_{\beta} \ln \mathcal{P}(\beta|x^{(i)}, y^{(i)}, 1 \leq i \leq N)$$

- Case 1: The prior distribution $\mathcal{P}(\beta_i = x) \propto \exp(-x^2)$ is Gaussian-like, and different β_i are independent. Now the minimization problem becomes

$$\min_{\beta} ||Y - \tilde{X}\beta||_2^2 + \lambda||\beta||_2^2.$$

here $||\beta||_2^2 = \sum_{i=0}^{p} \beta_i^2$. This is called **Ridge Regression**.

Here λ is the adjustable parameter in algorithm -- its choice is empirical while sometimes very important for model performance (where the word "alchemy" arises in machine learning!!!)

- Case 2: The prior distribution $\mathcal{P}(\beta_i = x) \propto \exp(-|x|)$ is double-exponential like, and different β_i are independent. Now the minimization problem becomes

$$\min_{\beta} ||Y - \tilde{X}\beta||_2^2 + \lambda \sum_{i=0}^{p} |\beta_i|$$

  This is called **LASSO Regression** (https://en.wikipedia.org/wiki/Lasso_(statistics)).

In general, these additional terms are called the **regularization terms**. In statistics, regularization is equivalent to Bayesian prior.

Algorithm consideration: The optimization for ridge regression is similar to OLS -- try to derive the analytical solution your self. The optimization for LASSO is non-trival (https://www.cs.ubc.ca/~schmidtm/Documents/2005_Notes_Lasso.pdf) and is the important topic in convex optimization.

Example: Diabetes Dataset

We use the scikit-learn package (https://scikit-learn.org/stable/index.html) to load the data and run regression. More tutorials about linear models can be found here (https://scikit-learn.org/stable/modules/linear_model.html).

Data from this paper (https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf) by Professor Robert Tibshirani et al (https://statweb.stanford.edu/~tibs/index.html).

In [ ]:

```
from sklearn import datasets
X,y= datasets.load_diabetes(return_X_y = True)
```

In [ ]:

```
help(datasets.load_diabetes)
```

Generate the training and test dataset by random splitting

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state
=42)
```

In [ ]:

```
print(X_train.shape)
print(y_test.shape)
```

In [ ]:

```
help(train_test_split)
```

Ordinary Least Square (OLS) Linear Regression

In [ ]:

```
from sklearn import linear_model
reg_ols = linear_model.LinearRegression()
reg_ols.fit(X_train,y_train)
```

In [ ]:

```
dir(reg_ols)
```

In [ ]:

```
reg_ols.coef_
```

In [ ]:

```
y_pred_ols = reg_ols.predict(X_test)
```

Evaluation:

- Mean Square Error (MSE) -- the lower, the better (in test data): $\frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$
- R-squared (coefficient of determination, R^2) -- the larger, the better (in test data): $1 - \frac{\sum_{i=1}^{N}(y^{(i)}-\hat{y}^{(i)})^2}{\sum_{i=1}^{N}(y^{(i)}-\bar{y})^2}$

In [ ]:

```
from sklearn.metrics import mean_squared_error
mse_ols = mean_squared_error(y_test, y_pred_ols)
R2_ols =  reg_ols.score(X_test,y_test)
print(mse_ols,R2_ols)
```

In [ ]:

```
reg_ridge = linear_model.Ridge(alpha=.02)
reg_ridge.fit(X_train,y_train)
print(reg_ridge.coef_)
y_pred_ridge = reg_ridge.predict(X_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
R2_ridge =  reg_ridge.score(X_test,y_test)
print(mse_ridge,R2_ridge)
```

In [ ]:

```
reg_lasso = linear_model.Lasso(alpha=.05)
reg_lasso.fit(X_train,y_train)
print(reg_lasso.coef_)

y_pred_lasso = reg_lasso.predict(X_test)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
R2_lasso =  reg_lasso.score(X_test,y_test)
print(mse_lasso,R2_lasso)
```

In [ ]:

```
print(reg_ols.score(X_train,y_train))
print(reg_ridge.score(X_train,y_train))
print(reg_lasso.score(X_train,y_train))
```

By definition, OLS has the smallest MSE (largest R-squared) on **training dataset**. What about on the test dataset?

In [ ]:

```
import numpy as np
train_errors = list()
test_errors = list()
alphas = np.logspace(-5, -1, 20)
for alpha in alphas:
    reg_lasso.set_params(alpha=alpha) # change the parameter of reg_lasso
    reg_lasso.fit(X_train, y_train)
    train_errors.append(reg_lasso.score(X_train, y_train))
    test_errors.append(reg_lasso.score(X_test, y_test))
```

In [ ]:

```
import matplotlib.pyplot as plt
fig = plt.figure(dpi=100)
plt.semilogx(alphas,train_errors,label = 'train R2')
plt.semilogx(alphas,test_errors,label = 'test R2')
plt.xlabel('alpha')
plt.legend()
```

Cross Validation (https://scikit-learn.org/stable/modules/cross_validation.html)

In [ ]:

```
from sklearn.model_selection import cross_val_score
scores_lasso = cross_val_score(reg_lasso, X, y, cv=10)
scores_ridge = cross_val_score(reg_ridge, X, y, cv=10)
scores_ols = cross_val_score(reg_ols, X, y, cv=10)
```

In [ ]:

```
print(scores_lasso)
print(scores_ridge)
print(scores_ols)
```

In [ ]:

```
help(cross_val_score)
```

In [ ]:

```
import pandas as pd
import seaborn as sns
scores_all = pd.DataFrame({"lasso": scores_lasso,"ols": scores_ols, "ridge":scores_ri
dge})
scores_all
```

Besides mean and standard deviation, we can also use the boxplot (https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51) to visualize the results.

In [ ]:

```
fig, ax = plt.subplots(dpi=100)
sns.boxplot(data = scores_all)
```