HSS 611 - Week 15: Machine Learning

2023-12-04

Agenda

- What is machine learning?
  - Supervised vs. unsupervised learning
- Fundamental concepts
  - Parameters and hyperparameters
  - Bias, variance, overfit
  - Train, test, (cross)validation
- `scikit-learn`
  - Linear regression (house price prediction)
  - Logistic regression (text classification)

What is machine learning?

Machine learning

- Enable computers "**machine**" to "**learn**" from data
- Algorithms and models
  - Algorithms: a sequence of instructions (or a computational procedure) that specifies how a task should be performed
  - Models: created by applying algorithms to data during through "training" by learning parameters from labeled or unlabeled data

What is machine learning?

Two fundamental approaches in machine learning

	Supervised	Unsupervised
Objective	Trained on a labeled data to learn a mapping from input to output	Find patterns or structures within data wihtout labeled data
Outcome	Pre-defined categories	Not quite pre-defined
Common tasks	Regression, classification	Clustering, dimensionality reduction
Model evaluation	Explicit metrics such as accuracy, precision, recall, or MSE	Can involve qualitative assessment

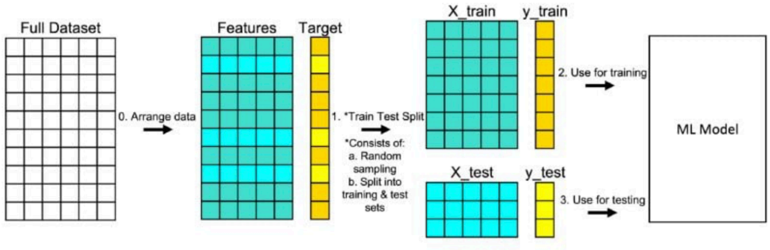
What is machine learning?

- Python is a really popular language in ML
- Traditional machine learning: scikit-learn
- Deep learning: PyTorch, Tensorflow & Keras

What is machine learning?

We will focus on **supervised learning**

- Involves a **training** and a **test** set
- Train a model using the **training** set
- Test the performance of the model on the **test** set



Fundamental concepts

Parameter and hyperparameters

- Parameter
  - Learned (estimated) from data (internal to the model)
  - E.g., regression weights/coefficients
- Hyper-parameters
  - Controls the learning process (thus hyper)Model structures,
  - Model structures (e..g, number of layers in NN), optimization approaches (e.g, learning rate in NN), etc.

Fundamental concepts

Bias

- The degree to which the model's predictions deviate from the actual values in a consistent manner
- A model with high bias tends to make predictions that are consistently off-target

Variance

- The degree to which the model generalizes to different data
- High variance means low generalizability

Fundamental concepts

Overfit

- If a model learns the training data “too well”, it can lead to **overfit**
- This happens when the model mistakes **noise** for **signal**
- The model will perform well on the training set but would not generalize to unseen data (i.e., test set)

Fundamental concepts

Validation

- A **validation set** is used to fine-tune hyperparameters

Fundamental concepts

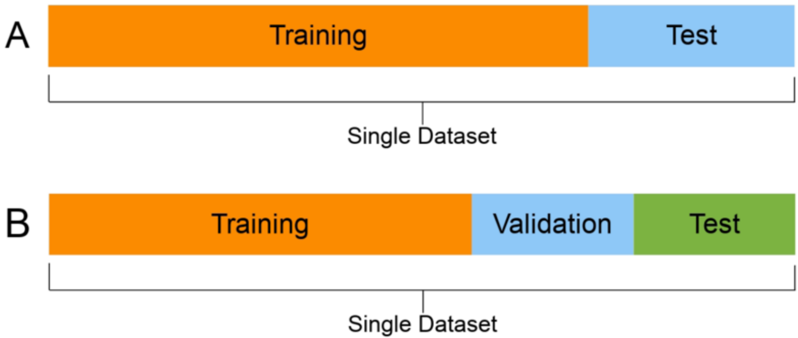


Figure 1: Train, Test, Validation sets (Wikimedia Commons)

Fundamental concepts

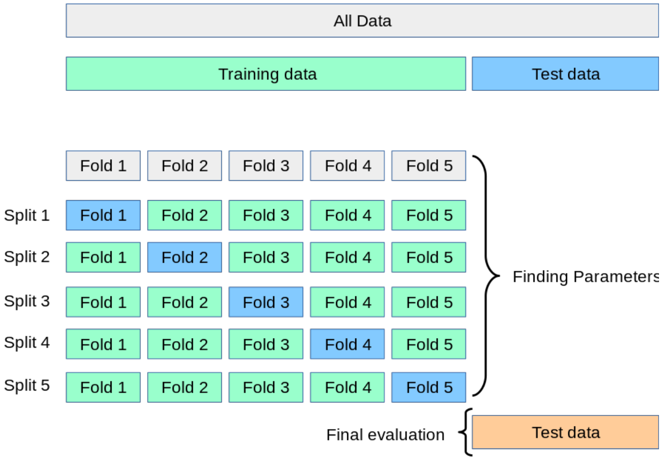


Figure 2: Cross Validation (scikit Learn)

Linear regression

- Most basic ML method for continuous outcome variables

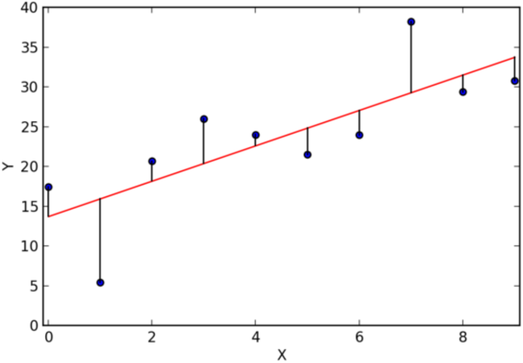


Figure 3: A linear regression model (Wikimedia Commons)

Ames housing data

- A dataset to **predict** house prices in Ames, Iowa
- Available through Kaggle
- We'll use it to apply some machine learning using `scikit-learn`

Load the Ames housing dataset

```
import pandas as pd
```

```
# urls
train_url = 'https://raw.githubusercontent.com/taegyoon-kim/programming_dhcss_2
test_url = 'https://raw.githubusercontent.com/taegyoon-kim/programming_dhcss_23

# load training set
train_df = pd.read_csv(train_url)

# load test set
test_df = pd.read_csv(test_url)

# number of observations
print(len(train_df))
print(len(test_df))
```

```
1460
1459
```

Inspect the training set

```python
train_df.columns
```

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

Inspect the test set

- The 'SalePrice' column in the test set is withheld by Kaggle

```
test_df.columns
```

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition'],
      dtype='object')
```

Select features

- The data set has a lot of features
- Let's use some of them to build a predictive model
- We can select them using a list

```
selected_features = ['FullBath', 'LotArea',
                        'OverallQual', 'YearBuilt']
target_variable = 'SalePrice'
```

Slice

```
X_train = train_df[selected_features]
y_train = train_df[target_variable]
```

```
print(X_train)
```

```
      FullBath  LotArea  OverallQual  YearBuilt
0            2     8450            7       2003
1            2     9600            6       1976
2            2    11250            7       2001
3            1     9550            7       1915
4            2    14260            8       2000
...        ...      ...          ...        ...
1455         2     7917            6       1999
1456         2    13175            6       1978
1457         2     9042            7       1941
1458         1     9717            5       1950
1459         1     9937            5       1965

[1460 rows x 4 columns]
```

Slice

```
print(y_train)
```

```
0       208500
1       181500
2       223500
3       140000
4       250000
         ...
1455    175000
1456    210000
1457    266500
1458    142125
1459    147500
Name: SalePrice, Length: 1460, dtype: int64
```

Fit model

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

```
LinearRegression()
```

```
model.coef_
```

```
array([2.07725220e+04, 1.40273782e+00, 3.70559379e+04, 2.20233453e+02])
```

- In a machine learning context, coefficients and standard errors are secondary
- Predictive performance is more important
- `scikit-learn` does not produce standard errors, p-values, confidence intervals, etc.
- See Shmueli 2010 for the differences between prediction and explanation

Make predictions

- Slice the test set in the same way

```
X_test = test_df[selected_features]
y_pred = model.predict(X_test)
y_pred
```

```
array([127736.09887712, 167841.57795785, 159534.27025935, ...,
       139268.00289048, 132906.70253888, 226869.50518936])
```

Add predictions to test set

- Now that we have predictions, we can add them to the test set

```
test_df['SalePrice'] = y_pred
test_df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotS
0	1461	20	RH	80.0	11622	Pave	NaN	Reg
1	1462	20	RL	81.0	14267	Pave	NaN	IR1
2	1463	60	RL	74.0	13830	Pave	NaN	IR1
3	1464	60	RL	78.0	9978	Pave	NaN	IR1
4	1465	120	RL	43.0	5005	Pave	NaN	IR1
...	...	...	...	...	...	...	...	...
1454	2915	160	RM	21.0	1936	Pave	NaN	Reg
1455	2916	160	RM	21.0	1894	Pave	NaN	Reg
1456	2917	20	RL	160.0	20000	Pave	NaN	Reg
1457	2918	85	RL	62.0	10441	Pave	NaN	Reg
1458	2919	60	RL	74.0	9627	Pave	NaN	Reg

Let's do better

- We can create and use a validation set
- Estimate what the performance of the model is going to be
- Adjust model based on that (e.g. add parameters, regularization, etc.)

Create validation set

- Create validation set from the training set

```
from sklearn.model_selection import train_test_split
```

- Approximately 20% of the observations will go to the validation set

```
X_train_small, X_val, y_train_small, y_val = train_test_split(
  X_train,
  y_train,
  test_size = 0.2,
  random_state = 7)
```

Train the model on the new training set

- Train on the new (smaller) training set

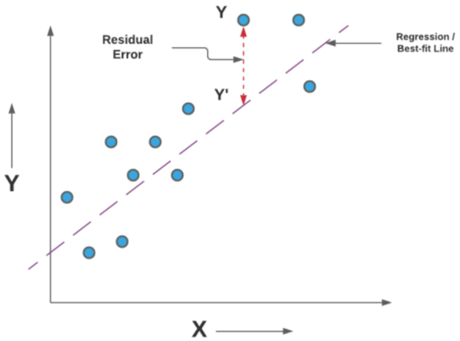
```
model.fit(X_train_small, y_train_small)
```

```
LinearRegression()
```

- Use the rest as the validation set; make prediction

```
y_val_pred = model.predict(X_val)
```

Predict Performance



$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

Where,

$\hat{y}$ – *predicted value of y*

$\bar{y}$ – *mean value of y*

Predict Performance

```
from sklearn.metrics import mean_squared_error
```

- Now, because we know the actual values, we can guess performance

```
import numpy as np
mse = mean_squared_error(y_val, y_val_pred) # MSE
np.sqrt(mse) # RMSE
```

```
48743.65433824328
```

Predict performance

```
from sklearn.metrics import r2_score
r2 = r2_score(y_val, y_val_pred)
r2
```

```
0.6714784029706313
```

K-fold cross-validation

- K-fold cross-validation usually a better method to estimate performance
- Not too sensitive to the randomness in the split
- Split data into **k** folds (usually 5 or 10)
  - Each fold takes turns being validation set
  - Each time, remaining folds serve as training set
  - Fit **k** models, predict for validation set
  - Estimate performance by averaging over all folds

K-fold cross-validation

- Import `cross_val_score`, create a `LinearRegression()` object (the latter not strictly necessary)

```
from sklearn.model_selection import cross_val_score
model = LinearRegression()
```

```
cv_scores = cross_val_score(model, X_train, y_train,
cv = 5, scoring = 'r2')
```

K-fold cross-validation

```
print(cv_scores)
print(np.std(cv_scores))
```

```
[0.70933564 0.65961618 0.66394196 0.67502642 0.6430035 ]
0.022116607595075617
```

Resources

- Introduction to Statistical Learning with Python (free!)
- Müller, A. C., & Guido, S. (2016). Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.".