Lecture 6-2

Preliminaries on Training ML Models

Contents

In the last lecture, we saw and trained some simple but useful ML models. In this short lecture, we will review key terminologies and outline the steps involved in training and inferencing a machine learning model.

1 Terminologies

Features and Labels

1. **Features** (Inputs), usually denoted by X (matrix form) or $\mathbf{x}$ (feature vector). These are the properties or characteristics of the data that the model uses as input.
   Example:
   - Direct representation of the molecular structure (1D, 2D, 3D representation)
   - Secondary representation: descriptors, e.g., molecular weight, number of rings, or fingerprints.
2. **Labels** (Targets/Outputs), usually denoted by y. These are the values or categories the model is trying to predict. Example: Toxic vs. non-toxic molecules, or a molecule's solubility.

Parameters and Hyperparameters

1. **Parameters**. These are the internal values of a model that are learned from the training data during the optimization process. They determine how the model maps inputs to outputs.
   Example: coefficients in a linear regression model.
2. **Hyperparameters**. These are external settings of the model or learning algorithm that are **fixed** before training and are not learned from the data. They control the training process and model complexity.
   Example: regularization strength.

Training and Inferencing

Machine Learning can be broadly divided into two distinct phases: the *learning* phase (training) and the *application* phase (inferencing).

1. **Training**. Use the training data to optimize ML model parameters θ, and use the training score (validation score) to tune hyperparameters.
2. **Inferencing**. Once an ML model is trained, it can be applied to new data to generate predictions.

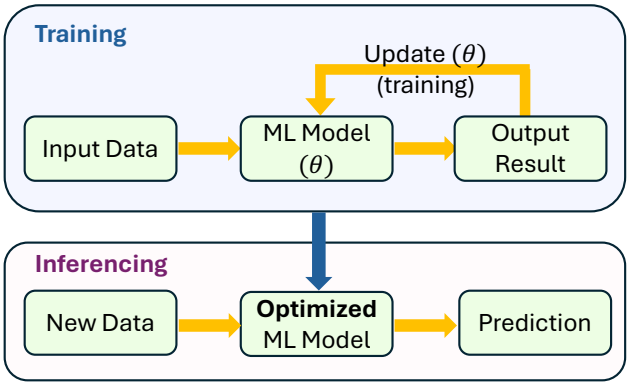


Figure 1: Training and inferencing.

Supervised and Unsupervised Learning

ML tasks can be broadly categorized based on the availability of labeled data: supervised learning and unsupervised learning.

1. **Supervised learning**. The training data are *labeled*, i.e., they consist of feature-label pairs $(\mathbf{x}, y)$, where y is the known label. The model learns to map inputs to outputs by minimizing the difference between its predictions and the true labels. Examples include regression and classification tasks.
2. **Unsupervised learning**. The training data contains only inputs x without any labels. The model tries to uncover patterns, structures, or groupings in the data. *Most* common generative models are trained on unlabeled data, so they fall under unsupervised learning.

Loss Function

The loss function (or cost function) is the mathematical target that quantifies the difference between ML predictions and the ground truth. ML model aims to minimize the loss function.

Example:

- mean squared error (MSE).
- cross-entropy.
- negative log-likelihood.

Example: mean squared error (MSE) for regression tasks, cross-entropy for classification tasks.

Train-Test Split

To evaluate the performance of a machine learning model, we typically divide the available data into two separate sets: the training set and the test set.

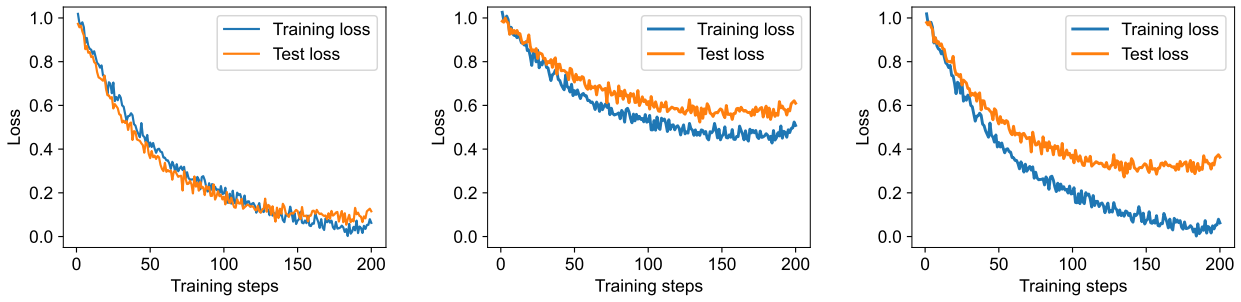
1. **Training set**. This subset of the data is used to train the model, i.e., to optimize its parameters and learn patterns from the data.
2. **Test set**. This separate subset is used to evaluate the model's performance on unseen data. It provides an estimate of how well the model generalizes to new inputs.

The test set is usually 20% of the whole set.

Underfitting and Overfitting

When training a machine learning model, it is important to balance complexity and generalization. Two common issues are underfitting and overfitting.

1. **Underfitting**. The model is too simple to capture the underlying patterns in the data. As a result, it performs poorly on both the training set and unseen test data.
2. **Overfitting**. The model is too complex and fits the training data too closely, including noise or random fluctuations. While it performs well on the training set, its performance on unseen test data is poor.

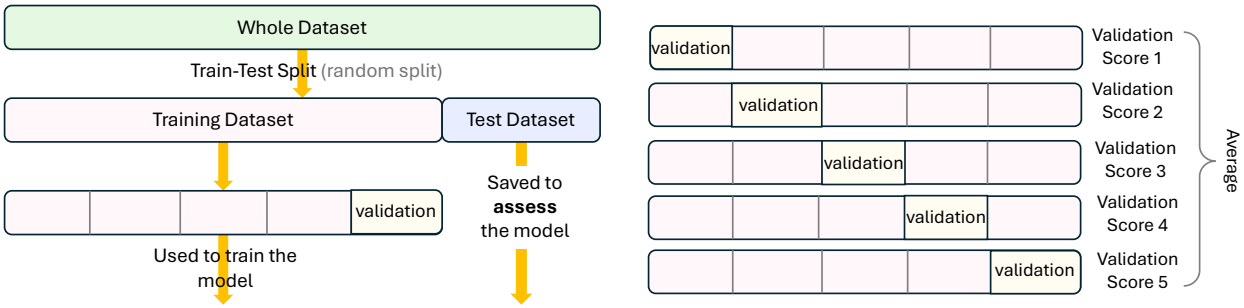


(a) Good Fitting. (b) Under Fitting. (c) Overfitting.

Figure 2: Examples of different fitting scenarios.

Validation Set and Cross-Validation

1. Validation Set. A separate subset of the training set used to tune hyperparameters and monitor model performance during training, helping to prevent overfitting.
2. Cross-Validation. A technique to estimate model performance more reliably by splitting the dataset into multiple train-validation folds and averaging the results across them.



(a) Train-test split and validation set.

(b) Cross validation.

Figure 3: Train-test split, validation set and cross validation.

2 Training and Inferencing Steps

We will use the linear regression and `sklearn` to illustrate the training and inferencing steps.

1. **Data Preparation.** Clean and preprocess the data, select features (e.g., rescaling + PCA).
   - The feature data X is usually represented as a $(n \times p)$ matrix, where n is the number of samples, and p is the number of features. So the rows are samples and columns are features.
   - The label data y is usually a vector of size n.
   - The above setting is used for major ML libraries such as `scikit-learn`, `TensorFlow` and `PyTorch`.
2. **Train-Test Split.** Split the features X and labels y into training and test sets.

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=None
)
```

3. **Model Selection.** Choose an appropriate model depending on the task (e.g., regression, classification, etc).

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

Some models have *hyperparameters*, and you can assign them when initiating the model, e.g.,

```
from sklearn.linear_model import Ridge
model = Ridge(alpha=0.01)
```

4. **Training the model.** Fit the model on the training data.

```
model.fit(X_train, y_train)
```

5. **Hyperparameter Tuning (Optional).** Adjust hyperparameters using a validation set or cross-validation to improve performance.
6. **Evaluation.** Use the test set to evaluate model performance and generalization.

```
from sklearn.metrics import mean_squared_error

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Test MSE:", mse)
```

7. **Inferencing.** Use the trained model to make predictions on new, unseen data. Save the model to a safe place.

```
y_new = model.predict(X_new)
```

8. **Saving the model.** We have not covered this part yet, but you can save a ML model with standard libararies, such as `joblib` and `pickle`.
   Joblib example:

```
import joblib

# Save model
joblib.dump(model, 'my_model.joblib')
# Load model
loaded_model = joblib.load('my_model.joblib')
```

   Pickle example:

```
import pickle

with open('model.pkl', 'wb') as f: # Save
```

```
    pickle.dump(model, f)
with open('model.pkl', 'rb') as f: # Load
    loaded_model = pickle.load(f)
```

Problems

1. Do we always have both features and labels in the training data?
2. Find the parameters and hyperparameters in the Ridge regression.

$$\text{Loss} = \sum_{i=1}^{n} (y_i - f_\theta(\mathbf{x}_i))^2 + \lambda \sum_{j=1}^{p} \theta_j^2,$$

where

$$f_\theta(\mathbf{x}_i) = \theta_0 + \sum_{j=1}^{p} \theta_j x_j.$$

3. Is your conversation with ChatGPT the training process or inferencing process? 4. Is linear regression supervised or unsupervised learning?