Seoul National University
College of Engineering
Department of Naval Architecture and Ocean Engineering
1, Gwanak-ro, Gwanak-gu, Seoul 151-744, Korea

Fall 2024

Machine Learning for Naval Architecture and Ocean Engineering

PA #1

Instructor name	김 태 완
Student name	이 하 빈
Department	조선해양공학과
Student ID	2020-17610
Submission date	24/10/13
Grade	

Programming Assignment 1

Habin Lee

October 13, 2024

Abstract

This report presents analysis of two fundamental machine learning concepts: polynomial regression and binary classification evaluation. In the first problem, we explore the bias-variance trade-off and overfitting phenomenon using polynomial regression on a noisy sinusoidal function. We visualize the relationship between model complexity and error, demonstrating how increasing polynomial degrees affects training and testing errors. The second problem focuses on evaluating a binary classification model using Receiver Operating Characteristic (ROC) and Precision-Recall (PR) curves. We generate a synthetic dataset, train a logistic regression model, and determine the optimal classification threshold using the F1 score. Through these experiments, we gain practical insights into model selection, performance evaluation, and the importance of balancing model complexity with generalization ability. Our findings highlight the critical role of visualization techniques in understanding and interpreting machine learning models.

Contents

1 Problem Definition Approach

1.1 Problem 1

Problem 1 is visualize overfitting and bias-variance trade-off using polynomial regression.

- Sample objective function with added noise
- Fit polynomial regression models of varying degrees
- Visualize overfitting
- Analyze bias-variance trade-off

1.2 Problem 2

Problem 2 is Evaluate binary classification model performance using ROC and PR curves, and find optimal threshold

- Generate binary classification dataset
- Train logistic regression model
- Create ROC and PR curves
- Determine optimal threshold using F1 score

2 Background Knowledge

2.1 Polynomial Regression

Polynomial regression is a form of regression analysis in which thee relationship between the independent variable x and dependent variable y is modeled as an nth degree polynomial in x. When β_i are coeffecients, n is the degree of the polynomial, ε is the error term, nth degree polynomial regression is like this

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n + \varepsilon$$

The goal is to find the coefficients that minimize the sum of squared residuals

$$\min \sum_{i=1}^{m} (y_i - (\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + ... + \beta_n x_i^n))^2$$

2.2 Overfitting

Overfitting occurs when a model learns the noise in the training data, leading to poor generalization on new, unseen data. Mathematically, we can express this as:

$$Error_{test} \gg Error_{train}$$

Where $Error_{test}$ is the error on the test set and $Error_{train}$ is the error on the training set.

2.3 Bias-Variance Trade-off

This concept explains the relationship between model complexity and generalization ability. The expected test error can be decomposed into three parts:

$$E[(y - \hat{f}(x))^2] = (Bias[\hat{f}(x)])^2 + Var[\hat{f}(x)] + \sigma^2$$

Where:

- $Bias[\hat{f}(x)]$ is the difference between the expected predictions and the true values
- $Var[\hat{f}(x)]$ is the variability of predictions
- σ^2 is the irreducible error

2.4 ROC Curve

The Receiver Operating Characteristic curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings:

$$TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN}$$

Where TP = True Positives, FN = False Negatives, FP = False Positives, TN = True Negatives.
The Area Under the Curve (AUC) quantifies the overall performance:

$$AUC = \int TPR \, d(FPR)$$

2.5 PR Curve

The Precision-Recall curve plots Precision against Recall:

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN}$$

This curve is particularly useful for imbalanced datasets. The Average Precision (AP) summarizes the curve:

$$AP = \int Precision \, d(Recall)$$

2.6 F1 Score

The F1 score is the harmonic mean of precision and recall, providing a single score that balances both metrics:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

It can be expressed in terms of TP, FP, and FN:

$$F1 = \frac{2TP}{2TP + FP + FN}$$

These concepts are fundamental in understanding model performance, especially in classification tasks. The trade-offs between these metrics often guide model selection and threshold tuning in practical applications.

3 Input/Output

3.1 Input

Problem 1

At requirement, We should determine the parameters through direct adjustment. So to approximate $sin(x) + \pi/2$ using polynomial regression in Problem 1, we need to determine the following parameters: total number of experiment repetitions(n_repeat), total sample size(n_sample), training data ratio(ratio), standard deviation of noise, and the range of polynomial degrees.

- n_repeat= The number of repetitions, determines how many times one 'train_set' is sampled. A sufficiently large number of repetitions (100) is chosen to enhance the stability and reliability of the experiment. This amount helps reduce the effects of randomness and provides more generalized results across multiple trials.
- n_sample: Total number of samples to extract.
- ratio: The proportion of the extracted samples to use as training data. We can utilize methods such as Hold-out, K-fold cross-validation, LOOCV, and Bootstrapping for model evaluation. However, considering the ratio variable among the example variables required in the Requirements, we believe

that using the Hold-out method with this ratio can allow for uniform sample extraction.
- n_train, n_test: The number of samples for training and testing, respectively, determined by ratio.
- noise: The range of noise (follows a normal distribution).

Problem 2

Binary classification dataset generation parameters - n_samples: This specifies the number of samples to generate. A larger sample size ensures that the model has enough data to learn meaningful patterns and generalize well.

-n_classes: This defines a binary classification problem, which is essential for evaluating the model using ROC and PR curves, as these metrics are designed for binary outcomes.
-n_features: The number of features determines the complexity of the dataset. Having multiple features allows for a more realistic and challenging problem, which is useful for testing the model's ability to handle high-dimensional data.
-test_size: Allocating 30% of the data for testing ensures that the model's performance can be evaluated on unseen data. This split provides a balance between training the model sufficiently and having enough data to test its generalization ability.
-random_state: Setting a random seed ensures reproducibility. This means that every time you run the code, you will get the same train-test split, allowing for consistent evaluation and comparison of results

3.2 Output

Problem 1

Figure 1 is overfitting graph. If you output a degree with a minimum test-mse, it is generally 11th order. After this 11th order, the difference between the test-mse and train-mse increases.

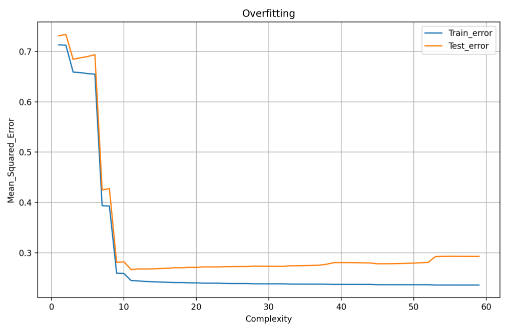


Figure 1: Overfitting Graph

Figure 2 illustrates the bias-variance trade-off and overfitting in a polynomial regression model. The blue line represents the bias squared, which initially remains low but slightly increases as the degree of the polynomial increases. This indicates that as the model becomes more complex, it starts fitting the noise in the data, leading to overfitting. The orange line shows the variance, which increases with higher model complexity, signifying that the model becomes more sensitive to fluctuations in the data. The green line represents the total error, calculated as the sum of bias squared, variance, and noise. Initially, the total error decreases with increasing complexity but starts to rise again, highlighting overfitting. The red dashed line indicates constant noise, representing unavoidable errors inherent in the data. The point where total error is minimized marks the optimal model complexity. Beyond this point, increasing variance leads to higher total error, demonstrating overfitting. This graph effectively visualizes the bias-variance trade-off and helps identify the optimal degree for minimizing error while avoiding overfitting, aligning with PA1's objectives in problem 1.

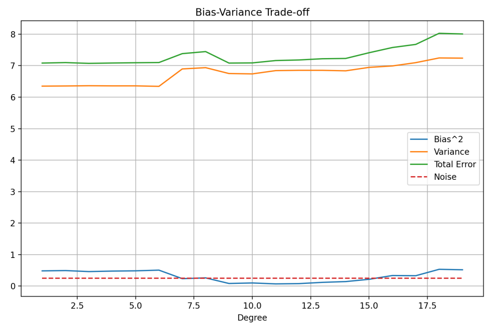


Figure 2: Bias-Variance trade-off graph

Figure 3 illustrates the relationship between the noisy samples, the true function, and the best-fit polynomial. The gray dots represent the noisy samples generated from the function $y = \sin(x) + x/2$ with added noise, simulating real-world data imperfections. The blue line depicts the true function, serving as the baseline for accuracy. The red line shows the polynomial fit of degree 11, which was chosen as it minimizes the mean squared error between the predicted and true values. This fit closely follows the true function while capturing variations in the noisy data, demonstrating a balance between fitting the data well and avoiding overfitting. This visualization highlights how a well-chosen polynomial degree can accurately model underlying patterns in noisy data, aligning with the objectives of PA1 Problem 1 by demonstrating overfitting.

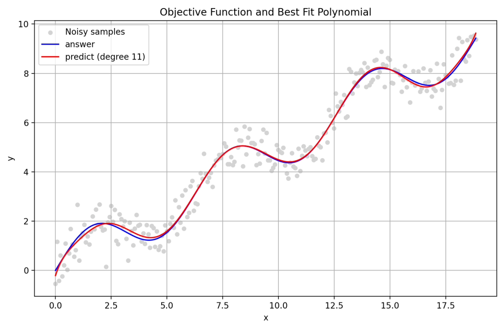


Figure 3: Optimal polynomial function graph

Problem 2

Figure 4 is a Receiver Operating Characteristic (ROC) curve, which illustrates the performance of a binary classification model. The curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. The area under the curve (AUC) is 0.91, indicating that the model has a high ability to distinguish between the two classes. A perfect model would have an AUC of 1.0, while a model with no discrimination ability would have an AUC of 0.5, represented by the diagonal line. The closer the ROC curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
Figure 5 is a Precision-Recall (PR) curve, which shows the trade-off between precision and recall for different threshold values. The average precision (AP) is 0.93, suggesting that the model maintains high precision across different levels of recall. This curve is particularly useful when dealing with imbalanced datasets, as it focuses on the performance of the positive class. The optimal threshold can be determined using the F1 score, which balances precision and recall, ensuring that both false positives and false negatives are minimized effectively. In this case, optimal threshold for PR curve is 0.4292.(Figure 6.)

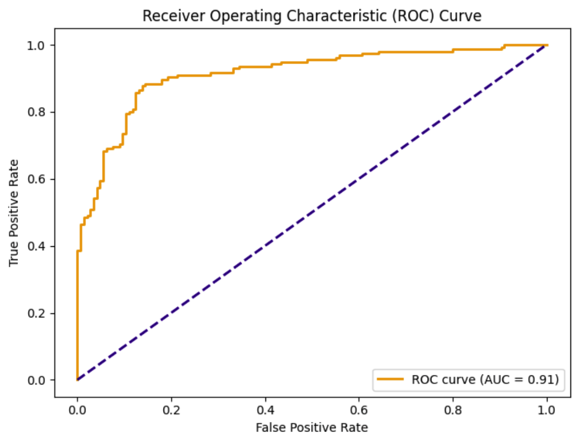


Figure 4: ROC curve

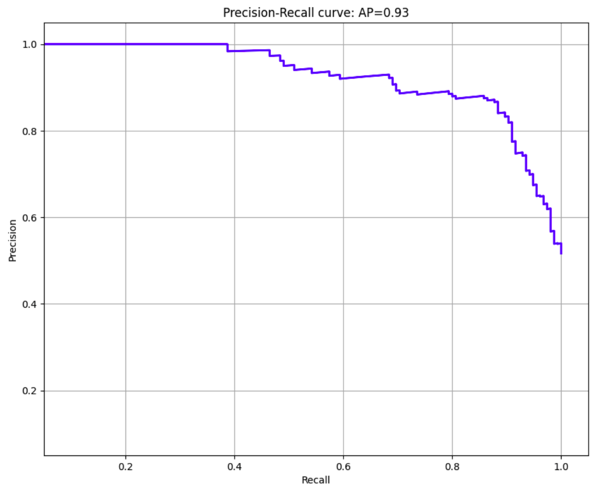
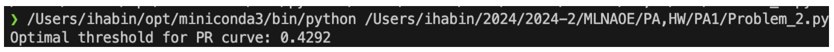


Figure 5: PR curve



```
❯ /Users/ihabin/opt/miniconda3/bin/python /Users/ihabin/2024/2024-2/MLNAOE/PA,HW/PA1/Problem_2.py
Optimal threshold for PR curve: 0.4292
```

Figure 6: Enter Caption

4 Code Analysis

4.1 Problem 1 Code

Functions

- `generate_objective_func`:

```
    def generate_objective_func(n_sample, noise_std):
        x = np.linspace(0, 6*np.pi, n_sample)
        normal_noise = np.random.normal(0, noise_std, n_sample)
        y = np.sin(x) + x/2 + normal_noise
        return x, y
```

  Generates a dataset based on the function $\sin(x) + \frac{x}{2}$ with added Gaussian noise.

- `fit_polynomial`:

```
def fit_polynomial(x, y, degree):
    return np.polyfit(x, y, degree)
```

Fits a polynomial of a specified degree to the given data using NumPy's `polyfit`.

- `MSE_Error`:

  ```
  def MSE_Error(y_true, y_predict):
      return np.mean((y_true - y_predict)**2)
  ```

  Calculates the mean squared error between true and predicted values.

Parameters

- `n_repeat`: Number of experiment repetitions to ensure stability of results.
- `n_sample`: Total number of data samples.
- `ratio`: Proportion of data used for training.
- `noise_std`: Standard deviation of noise added to the data.
- `degrees`: Range of polynomial degrees to evaluate.

Process

1. Generate data using the objective function with specified noise.
2. Shuffle and split data into training and test sets using:

   ```
   indices = np.random.permutation(n_sample)
   x_train, x_test = x[indices[:n_train]], x[indices[n_train:]]
   y_train, y_test = y[indices[:n_train]], y[indices[n_train:]]
   ```

3. For each polynomial degree, fit a model and predict test set outcomes:

   ```
   model = fit_polynomial(x_train, y_train, degree)
   y_test_pred = np.polyval(model, x_test)
   ```

4. Calculate bias, variance, and total error for each model:

   ```
   bias_squared = np.mean((y_true - y_test_pred) ** 2)
   variance = np.var(y_test_pred)
   total_error.append(bias_squared + variance + noise_std**2)
   ```

5. Repeat the process for a specified number of repetitions to average results.

Results The code calculates and plots:

- Average bias squared, variance, and total error across all repetitions.

  ```
  plt.plot(degrees, avg_biases, label='Bias^2')
  plt.plot(degrees, avg_variances, label='Variance')
  plt.plot(degrees, avg_total_errors, label='Total Error')
  plt.plot(degrees, [noise_std**2] * len(degrees), label='Noise', linestyle='--')
  ```

- These plots illustrate the bias-variance trade-off, showing how model complexity affects prediction error.

4.2 Problem 2 Code

Data Generation

```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_classes=2, n_features=20, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

A synthetic binary classification dataset is generated with 1000 samples and 20 features.

Model Training

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

A logistic regression model is trained on the training data.

Probability Prediction

```
y_pred_proba = model.predict_proba(X_test)[:, 1]
```

The model predicts probabilities for the test set.

ROC Curve

```
from sklearn.metrics import roc_curve, auc
fpr, tpr, roc_thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

The ROC curve is plotted to evaluate the trade-off between true positive rate and false positive rate. The AUC (Area Under Curve) is calculated to summarize the overall performance.

Precision-Recall Curve

```
from sklearn.metrics import precision_recall_curve, average_precision_score
precision, recall, pr_thresholds = precision_recall_curve(y_test, y_pred_proba)
average_precision = average_precision_score(y_test, y_pred_proba)

plt.figure(figsize=(10, 8))
plt.plot(recall, precision, color='b', lw=2)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
plt.grid(True)
plt.show()
```

The Precision-Recall curve is plotted to evaluate the balance between precision and recall. The average precision score is calculated as a summary metric.

Optimal Threshold Selection

```
f1_scores = 2 * (precision * recall) / (precision + recall)
optimal_idx = np.argmax(f1_scores)
optimal_threshold = pr_thresholds[optimal_idx]

print(f"Optimal threshold for PR curve: {optimal_threshold:.4f}")
```

The optimal threshold is determined using the F1 score to balance precision and recall.

5 Result

5.1 Problem 1: Bias-Variance Trade-off and Overfitting

In this problem, a polynomial function was fitted to sampled data to illustrate overfitting. As model complexity increased, both training and test errors initially decreased. However, beyond a certain point, the training error continued to decrease while the test error began to increase, indicating overfitting.

The total error was expressed as the sum of bias squared, variance, and noise. Changes in bias, variance, and error were visualized as model complexity increased. The polynomial function with the smallest error was identified and compared to the target function.

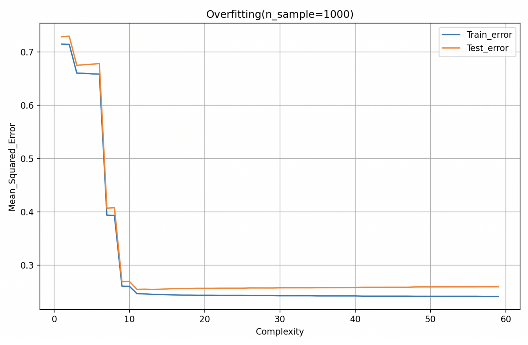
5.2 Problem 2: ROC and PR Curves

The Receiver Operating Characteristic (ROC) curve was plotted to show the relationship between the True Positive Rate and False Positive Rate. The Area Under the Curve (AUC) was 0.91, indicating strong model performance. The Precision-Recall (PR) curve demonstrated the trade-off between precision and recall across different thresholds. The Average Precision (AP) score was 0.93. Using the F1 score from the PR curve, the optimal threshold for classification was determined to be 0.4292. These analyses help in understanding model performance and selecting appropriate complexity and thresholds for optimal results.

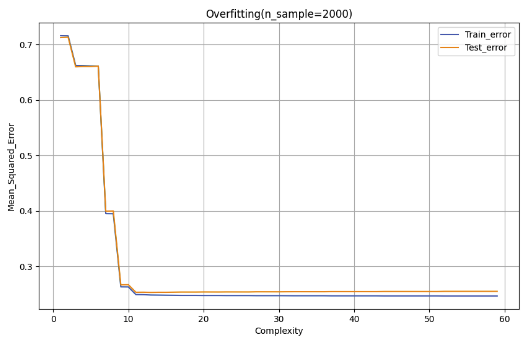
6 Discussion

6.1 How to reduce Overfitting

Increase sample size There are several methods to prevent overfitting, and one of the easiest is to increase the amount of data. In Problem 1, the value of 'n_sample' is set to 500, and we will check if increasing it can prevent ovefitting. In Figure 7, it can be observed that when 'n_sample' is 1000 and 2000, the MSE_error decreases in the degree range of 50 to 60.



(a) n_sample is 1000



(b) n_sample is 2000

Figure 7: Overfitting graph when n_sample increases

Cross-Validation Cross-validation involves splitting the dataset into multiple subsets and training/testing the model on these subsets. This process ensures that the model's performance is consistent across different data splits, reducing the likelihood of overfitting to a particular subset. It provides a more reliable estimate of model performance on unseen data. I modified the code with the following process to reduce overfitting to Cross-Validation

1. **Addition of K-fold Cross-Validation Function**:

- The function `cross_validate_polynomial` is defined to perform K-fold cross-validation on polynomial models of various degrees.
- It uses `cross_val_score` to calculate the Mean Squared Error (MSE) for each model.

2. **Storing Cross-Validation Results**:

- Within the main loop, the function `cross_validate_polynomial` is called for each iteration to perform cross-validation, and the results are stored in the list `all_cv_scores`.

3. **Calculating Averages and Outputting Optimal Degree**:

- The average of cross-validation scores from all iterations is calculated and stored in `avg_cv_scores`.
- The degree of the polynomial with the minimum average MSE is printed.

4. **Adding CV Error to Graph**:

- The cross-validation error (`CV_error`) is added to the graph to compare training error, test error, and cross-validation error.

The results shown in this process are shown in Figure 8 below. The difference between train_error and test_error decreased at degree range of 50 to 60 compared to Figure 1, but CV_error is increased at degree 20. In Figure 8, CV errors are increasing rapidly with increasing complexity, indicating that the model is learning noise from the data, a sign that overfitting is occurring. Even though CV is added, the reason why the CV_error rises rapidly from degree 20 and overfitting occurs is as follows.
- The Role of Cross-Verification: Cross-Verification is useful for assessing a model's generalization performance, but it does not prevent overfitting on its own. CV shows how well the model generalizes to different parts of the data, but does not reduce the overfitting of complex models.
- Model complexity: the high-order polynomial used in this code can learn up to noise in the data, leading to overfitting. CVs can detect this overfitting, but they do not directly prevent complex models from learning noise.
- Data Size and Noise: The data sample size is limited to 500, and noise is included, which can easily overfit high-order models. This shows an increase in CV errors.
- Normalization absent: no normalization technique was applied in this code. These techniques can help control model complexity and reduce overfitting.

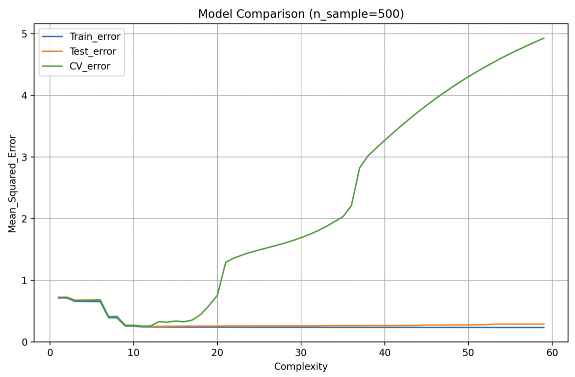


Figure 8: Overfitting with CV_error

Lasso Regression (L1 regularization) and Ridge Regression (L2 regularization)
Lasso Regression (L1 regularization) and Ridge Regression (L2 regularization) are techniques used to address overfitting in regression models, particularly for high-degree polynomials. Lasso Regression (L1 regularization) performs both variable selection and regularization. It can shrink some regression coefficients to exactly zero, effectively selecting a simpler model by excluding certain features. This property is particularly useful for high-degree polynomial models, where the risk of overfitting is significant due to the large number of predictors. Lasso encourages sparsity in the model, allowing identification and retention of only the most relevant variables. Ridge Regression (L2 regularization) adds a penalty term to the loss function, proportional to the sum of the squares of the coefficients. This helps minimize overfitting in polynomial models with high complexity. Unlike Lasso, Ridge Regression doesn't force coefficients to become exactly zero. Instead, it shrinks them towards zero, controlling their magnitude while retaining all predictors. Both techniques offer valuable tools for managing complexity in polynomial models. Lasso excels in feature selection, while Ridge provides a means to retain all predictors while controlling their impact on the model.

6.2 Limitations of ROC and PR Curves

ROC Curve Limitations

- **Misleading in Imbalanced Datasets**: ROC curves can be overly optimistic in imbalanced datasets for the following reasons:
  - They give equal weight to false positives and false negatives, which may not be appropriate in all scenarios.
  - The large number of true negatives in the majority class can lead to high specificity, making the model appear more effective than it actually is.
  - The performance on the minority class, often the class of interest, may be obscured.
- **Insensitivity to Class Distribution Changes**: ROC curves remain unchanged when the proportion of positive to negative instances varies, which can be problematic in real-world applications where class distributions may shift.
- **Lack of Intuitive Interpretation**: While AUC provides a single metric for model comparison, it doesn't offer intuitive insights into model behavior at specific operating points.

PR Curve Limitations

- **Sensitivity to Class Imbalance**: While this sensitivity can be beneficial, it also means that PR curves can vary significantly with changes in class distribution, making comparisons across different datasets challenging.
- **Lack of True Negative Representation**: PR curves do not account for true negatives, which can be important in some applications.
- **Difficulty in Comparing Models**: Unlike ROC curves, where a higher curve always indicates better performance, PR curves can intersect, making it less straightforward to compare models across all thresholds.

General Limitations for Both Curves

- **Threshold Dependency**: Both curves represent performance across all possible thresholds, which may not be practical in real-world scenarios where a single threshold must be chosen.
- **Aggregated Performance Metrics**: AUC and Average Precision summarize performance across all thresholds but don't provide information about performance at specific operating points.
- **Assumption of Fixed Costs**: Both curves assume that misclassification costs remain constant across all thresholds, which may not hold in many practical applications.

- **Limited to Binary Classification**: These curves are designed for binary classification problems and don't directly extend to multi-class scenarios.
- **Insensitivity to Prediction Confidence**: The curves don't distinguish between predictions made with high confidence and those made with low confidence, as long as they result in the same binary outcome.

To address these limitations, it's often beneficial to:

- Use both ROC and PR curves in conjunction, especially for imbalanced datasets.
- Consider alternative metrics or visualization techniques that are more appropriate for specific scenarios.
- Incorporate domain knowledge and specific business requirements when interpreting these curves and selecting optimal thresholds.
- Use additional evaluation methods such as calibration plots or decision curves for a more comprehensive model assessment.

7 Reference

Zhou, Z.-H. (2020). Machine Learning Foundations. (T. Kim, Trans.). Jpub.