Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant

Model Refinement

Improving model performance based on evaluation results

AGENDA

1. Understanding Model Training Errors
2. Techniques for Model Refinement
3. Advanced Refinement Techniques (Optional/Advanced)
4. Hands-On: Model Refinement Practice

If you were a superhero, what would your superpower be?

By the end of this session, students will be able to:

- Evaluate key performance metrics for model assessment.
- Identify and address issues of underfitting and overfitting in models.
- Apply a structured process of data preprocessing, hyperparameter tuning, and ensemble methods for model improvement.
- Confidently implement refinement techniques to boost model accuracy and robustness.

Evaluation Metrics Recap:

Core Metrics

- Accuracy: The proportion of correct predictions out of the total predictions. Often used when class distributions are balanced.
- Precision: The ratio of correctly predicted positive observations to the total predicted positives. Precision is critical when false positives are costly (e.g., spam detection).
- Recall: The ratio of correctly predicted positive observations to all observations in the actual class. Useful in cases where capturing all positives is crucial, like detecting

diseases.
- F1-Score: The harmonic mean of precision and recall. Balances both metrics, making it ideal when both false positives and false negatives are important.

Choosing the Right Metric It is important to select the right metric that ensures the model aligns with the real-world requirements of a given problem:

- **Classification**: Use AUC in credit fraud detection to balance detection rates with false positives.
- **Regression**: Use RMSE in forecasting where larger errors need penalization, such as in stock price predictions.

1. Understanding Model Training Errors

In machine learning, errors are an inevitable part of building and refining models.

Understanding where these errors come from and how to manage them is essential for creating effective models that perform well on new, unseen data.

When we train a machine learning model, we want it to be as accurate as possible—not just on the training data, but also on the data it has never seen before. This is where the generalization of a model comes into play.

Good generalization means the model can perform well on unseen data, and is not just memorizing the training examples.

Some of the most important model training errors are:

1. **Bias:**

Imagine you're trying to guess the height of a group of people based on their age.

If you always guess the same height for everyone, no matter what their age is, that would be a biased guess.

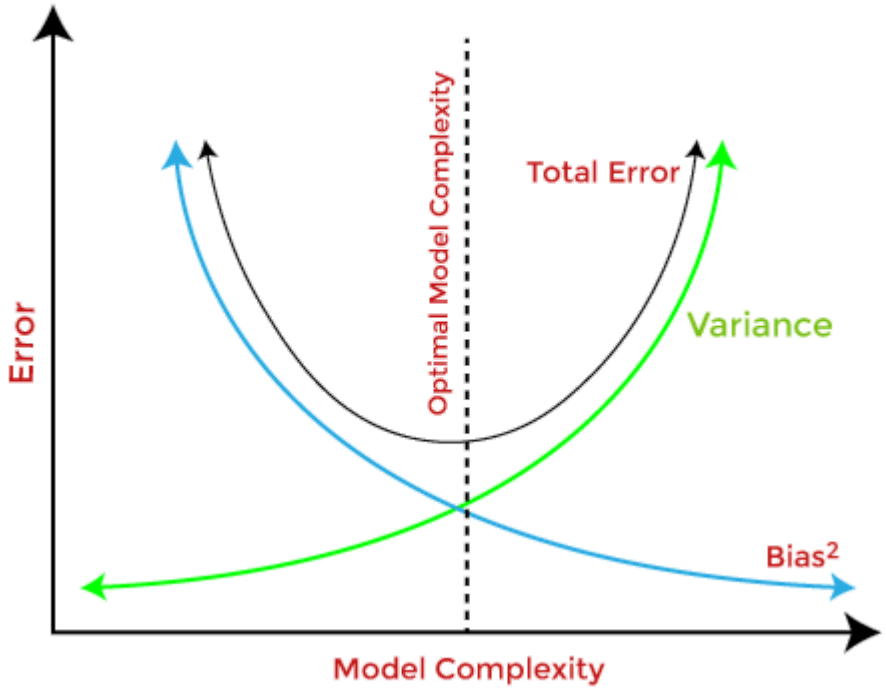
You're not considering their actual ages and just making a simple guess based on some wrong assumption.

Simple Example

Let's say you're using a machine learning model to predict house prices based on their size (in square feet). If your model always predicts the same price for every house, regardless of its size, that would be high bias. The model is too simple and doesn't take enough details into account. It's not learning the true relationship between house size and price.

- Bias refers to systematic errors in the model, introduced by approximating a real-world problem, which may be complex, by a simplified model.

- It occurs when the model makes assumptions that are too simple, often ignoring important features or relationships in the data.
- High bias typically results in underfitting, where the model is too simple to capture the underlying patterns of the data.
- Example: Using a linear regression model to predict house prices when the true relationship is non-linear (e.g., polynomial).



2. **Variance:**

Variance is like a model's sensitivity to small changes in the data.

Imagine you're trying to guess the height of people based on their age. If you adjust your guess every time you see a new person, even when they're about the same age, then your guesses might be all over the place — that's high variance!

In machine learning, high variance means the model is trying too hard to fit every little detail in the training data, even the noise or random patterns that aren't important.

Simple Example Suppose you have a model that predicts house prices based on the square footage and several other features. If your model is high in variance, it might make big adjustments for every single small change in these features. So, a house with just a tiny difference in square footage might get a drastically different price prediction, even if that change shouldn't really matter much.

- Variance refers to how much the model's predictions fluctuate when trained on different subsets of data.

- It occurs when the model is too complex and overly sensitive to the noise or fluctuations in the training data.
- High variance typically leads to overfitting, where the model fits the training data too well but fails to generalize to new, unseen data.
- Example: A decision tree with many branches that memorizes the training data instead of learning the general patterns.

So, you want to find a balance where the model is not too sensitive (high variance) but also not too simple (high bias). This helps the model perform well on both the training data and any new data it sees.

Bias-Variance Tradeoff

- The Bias-Variance Tradeoff refers to the tension between these two sources of error.

As you reduce bias (by using a more complex model), variance tends to increase (because the model is more sensitive to fluctuations in the training data), and vice versa.

- Goal: Find the optimal balance between bias and variance, where the model has both low bias and low variance, resulting in good generalization to new data.

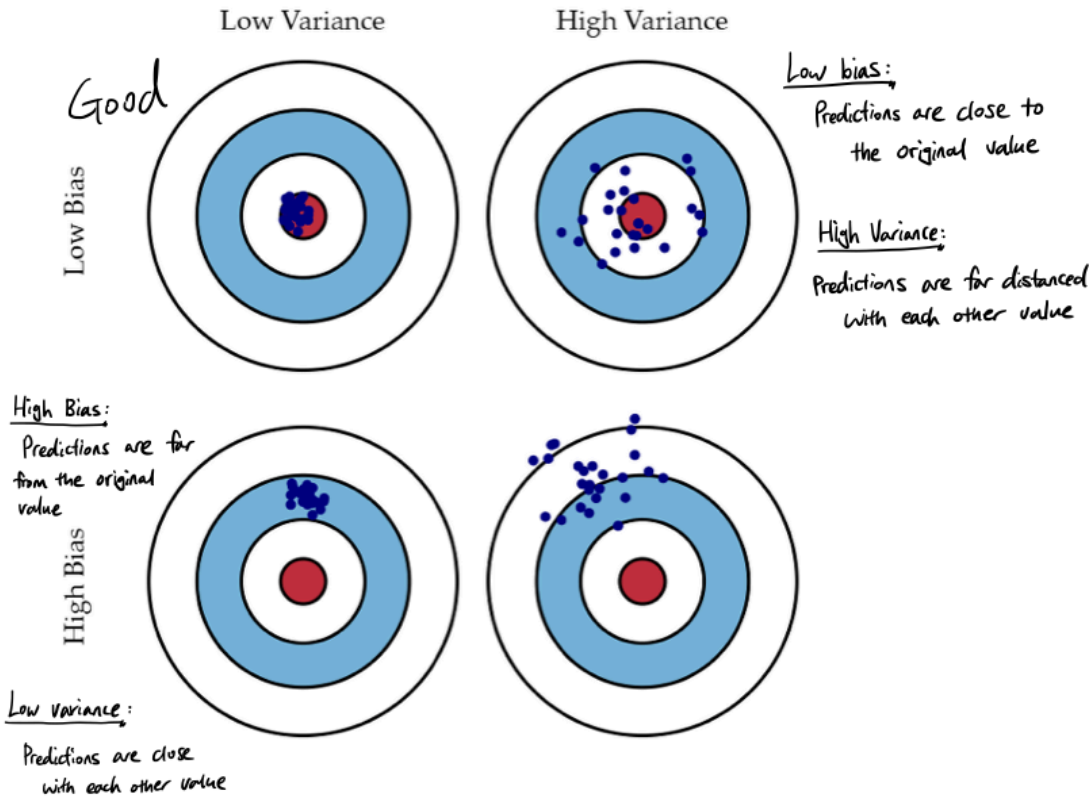


Fig. 1 Graphical illustration of bias and variance.

- Click here to learn more

3. Underfitting vs. Overfitting

Underfitting occurs when the model is too simple and cannot capture the underlying patterns in the data. This leads to poor performance on both the training set and the test set.

- High Bias, Low Variance: Simple models (underfit).

Overfitting occurs when the model is too complex and captures not just the true patterns but also the noise in the data. This leads to excellent performance on the training set but poor generalization to new, unseen data

- Low Bias, High Variance: Complex models (overfit).

4. Data Imbalance

Data imbalance happens when you have unequal numbers of examples in different classes of your dataset.

For example, if you're building a model to detect fraud, you might have a dataset where 95% of transactions are normal and only 5% are fraudulent. This difference is called imbalanced data.

- Bias Toward Majority Class: When data is imbalanced, the model may become biased towards the majority class (the larger group). So, in the fraud example, the model might mostly predict "normal" because it's the majority. This can cause the model to ignore or perform poorly on the minority class (fraudulent transactions).

- Poor Accuracy on on Minority Class: If the minority class (fraud) is more important to predict, imbalance can be especially problematic. Even if the model achieves high overall accuracy by predicting the majority class, it might fail to detect the minority class, which is often the class we care most about.

In real life, we often fix data imbalance by resampling the data

— either by **oversampling** the minority class or **undersampling** the majority class

— and instead of just accuracy, use metrics like Precision, Recall, and the F1 score, which give a better picture of how well the model is performing on each class.

Additionally, special algorithms like SMOTE (Synthetic Minority Over-sampling Technique) can create synthetic samples to help balance the dataset.

5. Data Drift

Data Drift happens when the data used to train a model changes over time, so it no longer matches the new data the model sees in the real world. This can lead to poor performance because the model is making predictions based on outdated patterns.

Reasons:

- Changing User Behavior: For example, in a recommendation system, user preferences might shift over time.
- Seasonal Changes: Data may change with the seasons or economic conditions (like shopping trends during holidays).
- System Updates: Changes in how data is collected or processed (like updates to sensors or apps) can also cause drift.

Types of Data Drift:

1. **Covariate Drift:** When the input data (features) distribution changes but the relationship to the target variable remains the same.

Example: Let's say a model was trained to assess creditworthiness based on features like age, income, and spending habits.

Over time, if economic conditions change (like during a recession), people's spending habits and income levels may shift, even though the relationship between these features and credit risk (the target variable) remains the same.

2. **Concept Drift:** When the relationship between inputs and the target variable itself changes.

Example: Suppose a model is trained to classify emails as spam based on specific keywords and patterns. Over time, as spammers develop new tactics, the patterns in spam emails change (e.g., new words, different structures).

Now, the original relationship between certain keywords and the label "spam" no longer holds, meaning the model becomes less effective. This change in the underlying relationship between inputs (keywords) and the target (spam/not spam) is concept drift.

In real life, we handle data drift by regularly monitoring model performance, retraining the model with updated data, and using adaptive models that adjust to new data patterns.

Model Refinement

Model Refinement in machine learning refers to the process of improving a model's performance by iterating on it through various techniques.

This step is crucial to ensure that the model generalizes well to new, unseen data

The model refinement process typically includes:

1. **Evaluating Model Performance:** Carefully analyzing model metrics to identify where the model is falling short.
2. **Identifying Key Issues:** Pinpointing problems like high bias (indicating underfitting), high variance (overfitting), or data imbalance.
3. **Applying Improvement Techniques:** Using targeted methods, such as tuning hyperparameters, changing model architecture, or refining the dataset, to tackle

identified issues.

4. **Iterative Testing and Validation:** Running and evaluating improvements in multiple rounds to ensure they lead to better and more stable results.

2. Techniques for Model Refinement

Some Techniques to systematically refine your models and make their outputs and predictions better.

Step 1: Data Preprocessing Refinement

In this first step of model refinement, we'll focus on preparing the data to improve model performance. Well-preprocessed data ensures that models learn effectively, reducing bias and variance in the results.

Data Scaling and Normalization:

In scaling (Standardization), the goal is to standardize features so they have a mean of 0 and standard deviation of 1

- **When to Scale:** Scaling techniques like StandardScaler or MinMaxScaler are critical when using algorithms sensitive to feature magnitudes, such as distance-based algorithms (e.g., K-Nearest Neighbors) and linear models.
- **How it Helps:** Scaling or normalizing features to the same range or distribution can improve model convergence, stability, and predictive performance.

In Normalization, the goal is to transforms features so they fall within a certain range, typically [0, 1] or [-1, 1].

More or less the same, but normalization **adjusts the range of data** to a specific interval (e.g., [0, 1]), while **scaling adjusts the spread** or **distribution** of the data, typically by changing the mean and standard deviation.

Example

Suppose we have a dataset with the following features to predict house prices:

- **Size** (in square feet)
- **Number of Rooms**
- **Distance to City Center** (in miles)

Here's a sample of what the dataset might look like:

Size (sq ft)	Number of Rooms	Distance to City Center (miles)	Price ($)
2500	4	8	500,000

Size (sq ft)	Number of Rooms	Distance to City Center (miles)	Price ($)
1500	3	15	300,000
1200	2	20	200,000
1800	3	10	350,000

Without scaling, the large range of values (e.g., Size in thousands vs. Distance in tens) may lead some models to interpret Size as more important than other features. Here's how scaling can address this:

Standard Scaling

Standard scaling (using `StandardScaler`) adjusts each feature so that it has a mean of 0 and a standard deviation of 1.

Transformed dataset (Standard Scaled):

Size (scaled)	Number of Rooms (scaled)	Distance to City Center (scaled)
1.14	1.0	-0.6
-0.2	0.0	0.9
-0.8	-1.0	1.5
-0.1	0.0	-0.3

Min-Max Scaling

Min-Max scaling (using `MinMaxScaler`) transforms each feature to a specified range, typically [0, 1].

Transformed dataset (Min-Max Scaled):

Size (scaled)	Number of Rooms (scaled)	Distance to City Center (scaled)
1.0	1.0	0.0
0.3	0.5	0.54
0.0	0.0	1.0
0.5	0.5	0.31

Why This Matters

- **Improved Model Performance**: Models sensitive to feature scales, like K-Nearest Neighbors and Linear Regression, perform better with scaled features.
- **Faster Convergence**: Gradient-based models, like logistic regression or neural networks, converge faster when features are on a similar scale.

Find code implementation in the "DataScaling and Normalisation.ipynb"

Techniques Used in Scaling

- Standard Scaling (Z-score Normalization)
- Robust Scaling
- Max Abs Scaling

Techniques Used in Normalization

- Min-Max Normalization (Rescaling)
- L2 Normalization (Euclidean norm)
- L1 Normalization (Manhattan norm)

Feature Engineering:

- **Feature Selection:** Removing features with high correlation or low variance can reduce redundancy and prevent the model from overemphasizing irrelevant or repetitive information.
- **Feature Creation/Extraction:** New features can be engineered by transforming or combining existing ones, potentially uncovering valuable patterns that the model might otherwise miss.

Handling Class Imbalance:

- **Why it Matters:** In classification problems, class imbalance can cause models to perform poorly on the minority class, as they may simply predict the majority class to achieve high accuracy.
- **Techniques to Address Imbalance:**
  - **Oversampling** (e.g., duplicating instances of the minority class).
  - **Undersampling** (e.g., reducing the majority class).
  - **SMOTE (Synthetic Minority Over-sampling Technique)**, which creates synthetic samples for the minority class.

Step 2: Hyperparameter Tuning

Hyperparameters are configurations **external to the model** that cannot be learned from data during training. They differ from parameters, which are learned by the model.

For instance, in a **decision tree**, **parameters** include **split points** within nodes, whereas **hyperparameters** may include the **tree depth** or **minimum number of samples** required to split a node.

Proper tuning of hyperparameters can significantly enhance the model's accuracy, generalizability, and training efficiency.

Hyperparameters vs. Parameters

- **Parameters**: Learned directly from the data (e.g., weights in a linear regression model).

- **Hyperparameters**: Set manually before training begins and control the behavior of the learning algorithm (e.g., k in k-nearest neighbors).

Hyperparametes for some commonly used algorithms

1. **Linear Regression**
    - **Hyperparameters**:
        - Regularization strength (for Ridge and Lasso: alpha)
        - Solver type (e.g., 'lbfgs', 'saga')
2. **Logistic Regression**
    - **Hyperparameters**:
        - Regularization strength (C)
        - Solver type (e.g., 'liblinear', 'newton-cg')
        - Max iterations
3. **Decision Trees**
    - **Hyperparameters**:
        - Maximum depth (max_depth)
        - Minimum samples split (min_samples_split)
        - Minimum samples leaf (min_samples_leaf)
        - Max features
4. **Random Forest**
    - **Hyperparameters**:
        - Number of trees (n_estimators)
        - Max depth (max_depth)
        - Max features
        - Minimum samples per leaf (min_samples_leaf)
        - Bootstrap (for sampling)
5. **Support Vector Machines (SVM)**
    - **Hyperparameters**:
        - Regularization parameter (C)
        - Kernel type (e.g., 'linear', 'rbf', 'poly')
        - Gamma (for RBF kernel)
        - Degree (for polynomial kernel)
6. **K-Nearest Neighbors (KNN)**
    - **Hyperparameters**:
        - Number of neighbors (n_neighbors)
        - Distance metric (e.g., 'euclidean', 'manhattan')
        - Weights (uniform or distance)
7. **Gradient Boosting Machines (GBM) / XGBoost / LightGBM**
    - **Hyperparameters**:
        - Number of trees (n_estimators)

    - Learning rate (learning_rate)
    - Maximum depth (max_depth)
    - Subsample
    - Min child weight (XGBoost)
8. **Naive Bayes**

    - **Hyperparameters**:
        - Smoothing parameter (alpha)
9. **K-Means Clustering**

    - **Hyperparameters**:
        - Number of clusters (n_clusters)
        - Initialization method (init: 'k-means++' or 'random')
        - Max iterations (max_iter)
        - Tolerance for convergence (tol)
10. **Neural Networks (e.g., MLPClassifier)**

    - **Hyperparameters**:
        - Number of hidden layers (hidden_layer_sizes)
        - Number of neurons per layer
        - Activation function (e.g., 'relu', 'tanh')
        - Learning rate (learning_rate)
        - Batch size
        - Epochs (max_iter)
        - Solver type (e.g., 'adam', 'sgd')
11. **Principal Component Analysis (PCA)**

    - **Hyperparameters**:
        - Number of components (n_components)
        - Whitening (True/False)

Common Hyperparameter Tuning Techniques

1. Grid Search:

This method tests all possible combinations of a predefined set of hyperparameter values exhaustively. Though powerful, it can be computationally expensive.

Example: Testing combinations of max_depth and min_samples_split for a decision tree.

2. Random Search:

Instead of testing all combinations, random search samples a fixed number of hyperparameter combinations randomly from a specified distribution. This often finds optimal settings more efficiently than grid search.

Example: Randomly selecting learning_rate and n_estimators values for a gradient boosting model.

3. Advanced Refinement Techniques (Optional/Advanced)

Ensemble Methods

- **Ensemble learning** refers to the technique of combining multiple individual models to create a stronger, more accurate predictive model.
- The goal is to leverage the strengths of different models while mitigating their weaknesses.
- By combining models, we can often achieve better generalization and performance than any single model in isolation.

Types of Ensemble Methods:

1. **Bagging (Bootstrap Aggregating):**
    - Involves training multiple models on different random subsets of the training data (with replacement) and combining their predictions.
    - Example: Random Forest, where multiple decision trees are trained and their predictions are averaged (regression) or voted on (classification).
2. **Boosting:**
    - Involves training models sequentially, where each new model tries to correct the errors of the previous one by focusing on misclassified instances.
    - Examples:
        - **AdaBoost**: Adjusts the weight of misclassified instances to improve subsequent models.
        - **Gradient Boosting**: Trains models in a sequential manner, where each model reduces the residual errors from previous ones.
        - **XGBoost**: An optimized version of gradient boosting, known for its efficiency and speed.
3. **Stacking (Stacked Generalization)**:
    - Involves training multiple models of different types (e.g., decision trees, SVMs) and combining their predictions using a final model (meta-model) that learns how to best combine them.
    - Example: A model that combines the predictions of a decision tree, a support vector machine, and a neural network into a final prediction using a logistic regression or another classifier as the meta-model.

Cross-Validation and Model Validation Techniques

K-Fold Cross-Validation:

- Cross-validation helps ensure that model performance is not overestimated by testing it on different subsets of data, rather than just one train-test split. This technique provides a more reliable estimate of model performance.

Advanced Cross-Validation (e.g., Stratified K-Fold, Time Series Split):

- **Stratified K-Fold**: This variation ensures that each fold of the data has a proportionate representation of the target classes, which is especially important for imbalanced datasets.
- **Time Series Split**: Used for time-dependent data where the order of the data is crucial (e.g., stock prices, weather data). This method avoids future data being used to predict past data, which is crucial for time-based datasets.

Train-Test-Split Best Practices:

- **Recap**: Emphasize the importance of a proper **train-test split**. The split should be done randomly to ensure diversity, but students should also be cautious of **data leakage**, which occurs when information from outside the training dataset is used in the model. This would lead to overly optimistic performance estimates.

Conclusion

- **K-Fold Cross-Validation** ensures that the model is tested multiple times on different subsets of the data, providing a more robust performance estimate.
- **Advanced Techniques** like **Stratified K-Fold** and **Time Series Split** help tackle challenges posed by imbalanced datasets or time-sensitive data.
- **Proper Train-Test Split** practices should be followed to avoid overfitting and data leakage, ensuring that models are validated correctly.

4. Hands-On Mini-Project: Model Refinement Practice

Project Title: Customer Churn Prediction Model Refinement Challenge

Description

This project centers on **predicting customer churn**, a popular real-world problem in industries like **telecom**, **SaaS**, or **e-commerce**. The dataset contains information about customer demographics, subscription details, service usage, and customer support interactions. You will build a machine learning model to predict whether a customer will churn (i.e., stop using the service) based on these features.

This project is modern, widely applicable, and gives you insight into solving a business-critical problem.

Instructions

1. **Define the Problem**: Predict customer churn using the Telco Customer Churn dataset.
2. **Data Preprocessing**: Handle missing values, encode categorical features, and scale numerical data.
3. **Baseline Model**: Train a simple classifier with minimal preprocessing.
4. **Evaluate Baseline**: Analyze accuracy, recall, and other key metrics.

Then attempt improve the metrics of the model.
5. **Feature Engineering**: Create new features, remove redundant ones, and analyze correlations.
6. **Hyperparameter Tuning**: Optimize model parameters using GridSearchCV or similar tools.
7. **Model Ensembling**: Use ensemble techniques like Gradient Boosting or Random Forest to enhance performance.
8. **Final Evaluation**: Compare refined model metrics with baseline results and assess business value.
9. **Presentation**: Summarize findings, improvements, and rationale for techniques used.

1. topic:
    - question

In [ ]:

Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant