

Lecture 1: Trade-offs and model training

Instructor: Justin J. Boutilier

===

In this lecture, we will introduce some important concepts that apply to most machine learning applications. We will learn:

1. The prediction-explanation trade-off
2. The bias-variance trade-off
3. How to select features and reduce overfitting
4. How to train and test models

*Corrections provided by Oliver Li, Jin-ri Lee, Luiz Gosling, and Jeffrey Maloney.

Prediction vs. Explanation

Machine learning problems typically fall into one of two categories based on our end goal:

- 1. Prediction:** Our goal is to predict the target value as accurately as possible for future observations. Because our focus is on prediction accuracy, fewer assumptions can be made at the cost of regression coefficient interpretation.
- 2. Explanation:** Our goal is to explain the relationship between the target and the features. In this case, we need to interpret the regression coefficients and care is needed with respect to model assumptions, otherwise the information we extract from the regression coefficients (e.g., statistical significance) may be misleading (at best) or incorrect (at worst).

For most problems, our end goal determines the type of models that we should employ. On the explanation end, classical regression models (e.g., linear and logistic regression) are often used because the regression coefficients provide detailed information (including statistical significance) about the relationship between each feature and the target. On prediction end, advanced models (e.g., neural networks, ensemble methods) are often used because they can capture more complex relationships, at the cost of limited information regarding the importance of each feature. Some models (e.g., decision trees, random forest) lie in the middle because they can capture complex relationships, while providing basic information about feature importance. For example, a random forest model provides us with a relative ranking of features according to their importance (we will learn more about this later!). Note that much of machine learning is focused on prediction problems.

Bias vs. Variance

For prediction problems, our focus is to obtain the most accurate out-of-sample model performance. In other words, we want our model to perform well on observations that it has not yet seen. To help build intuition, imagine that there are multiple groups of observations (i.e., multiple datasets) that we have not seen. When we apply our model to these unseen datasets, we will obtain an estimate of our out-of-sample prediction error (for each unseen dataset). We can decompose this error into three components of interest:

1. **Bias:** the average prediction error over all unseen datasets. Intuitively, bias measures the error caused by modelling assumptions that lead us to miss key relationships (e.g., wrong type of model, missing key features, etc.). A model with high bias is said to be *underfitting* the data.
2. **Variance:** the variation in prediction error over all unseen datasets. Intuitively, variance measures the model's sensitivity to a particular choice of dataset. A model with high variance is said to be *overfitting* the data.
3. **Irreducible error:** the prediction error due to uncontrollable factors. Intuitively, irreducible error is caused by measurement error, randomness, or other unforeseen factors.

Figure 1 provides a visualization of bias and variance. Ideally, we want a model with both low bias and low variance (upper left), meaning that it can capture the relationships in the training data and generalize well to unseen data. However, it can be challenging to simultaneously reduce both bias (underfitting) and variance (overfitting).

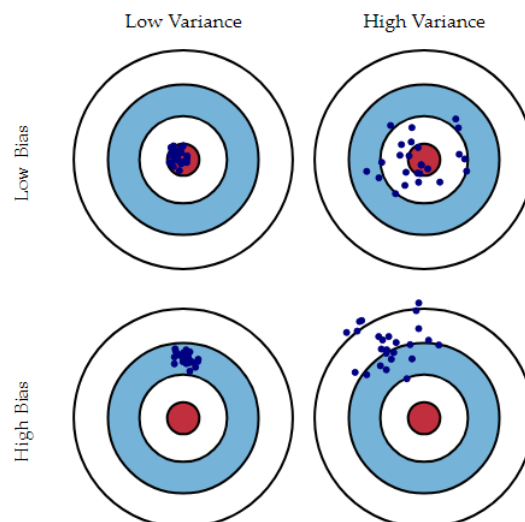


Figure 1: A visualization of bias and variance. Each dot represents the performance of our model on an unseen data set. Dots in the bulls-eye (i.e., the red circle) are most accurate. [Source](#).

It turns out that we can mathematically prove that out-of-sample error can be decomposed into the three aforementioned components. Recall that for regression problems y_i

is our target variable and \mathbf{x}_i is our feature vector. As a modeler, we assume that there exists a function f that maps the feature vector to the target: $y_i = f(\mathbf{x}_i) + \epsilon$, where ϵ represents the noise with zero mean and variance σ^2 . Our goal is to find a function or model \hat{f} that best approximates f . Note that different models will lead us to different \hat{f} . No matter what model we select, we can decompose the expected out-of-sample squared error for a particular dataset into:

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

where the first term represents the bias squared, the second term represents the variance, and the third term is the irreducible error caused by noise. Mathematically, the bias term represents the difference between the best possible model and the expected “long-term” performance of our model. The variance represents the difference between the expected “long-term” performance of our model and the performance of our model on a particular dataset.

A common cause of overfitting (or increased variance) is model complexity. For example, highly complex models (i.e., models with many features, models that capture non-linear relationships) can easily overfit small datasets by learning the random noise within the data. Figure 2 provides a conceptual visualization of error as a function of model complexity. Given our focus on prediction problems, the notion of overfitting will be a reoccurring theme throughout this course. Please see Section 3.2 in Bishop for the mathematical details of bias and variance. You can find examples of the bias-variance trade-off [here](#).

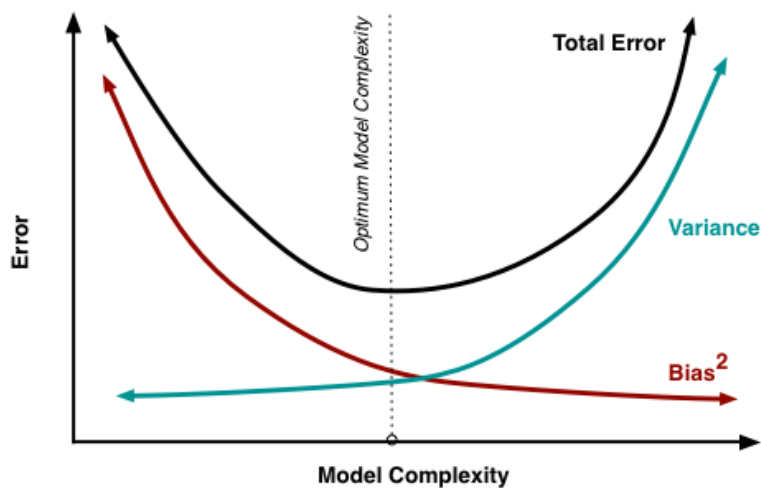


Figure 2: A visualization of bias, variance, and total error as a function of model complexity. [Source](#).

Feature Selection

Feature selection is the process of identifying which features are relevant in predicting the target and therefore need to be included in the model. Feature selection is (one of)

the most challenging aspects of machine learning and to this day, there is no universally accepted gold standard.

- **Correlated features:** The first step is to look for and remove highly correlated features (i.e., correlation coefficient above 0.8 or below -0.8). We remove them because they provide very similar signals and may even represent the same underlying relationship. It is especially important to remove correlations for explanation problems, because correlated features can bias regression coefficients. Correlated features should also be removed from prediction problems because they can impact the numerical algorithms used to train the model. Although we can easily detect correlated features, it can be challenging to determine which of the correlated features we should remove. This is typically a tedious manual process, but a simple rule of thumb is to remove the minimum number of features, such that there are no more correlations.

For applications with small feature sets and/or where the modeller has some domain specific knowledge, a manual approach for feature selection may be appropriate:

- **Expert opinion:** The modeller and/or an expert in the field manually chooses the features to include based off previous literature, hypothesized relationships, and/or intuition. This approach is particularly useful for explanation problems that seek to test hypotheses (or relationships), while controlling for other features.

For applications with large feature sets and/or where the modeller has no domain specific knowledge, an automated approach for feature selection is needed:

- **Exhaustive search:** Compare model performance for all possible feature combinations and select the best. This is often only possible for small feature sets due to computational complexity.
- **Stepwise:** A commonly used approach for feature selection which incrementally adds or removes features. Stepwise methods should not be used for explanation problems because they can significantly bias the regression coefficients and corresponding statistical tests. See [here](#) and [here](#) for more information.
- **Regularization:** An automated approach for reducing overfitting that has demonstrated effectiveness as a feature selection method (see the next section!).

Regularization

Regularization was developed as a method to reduce overfitting by penalizing the regression coefficients. Intuitively, regularization mitigates overfitting by reducing the model complexity. A specific type of regularization can also be used as an automated approach for feature selection. In regularized regression, we add an additional term, $\lambda \sum_{k=1}^K |\beta_k|^p$, to the objective of the least squares problem, where p is a number that denotes the type of regularization and λ is a hyperparameter. The λ parameter controls the degree of regularization and can be chosen through a cross-validation procedure (more on this below!). Intuitively, you can think of regularization as penalizing non-zero regression coefficients,

where different types of regularization impose different types of penalties. Regularization is useful because it limits the size of the coefficients, preventing over-fitting to the training data. There are three common types of regularization:

- **L1-regularization** ($p = 1$): Also called **lasso** regression. Lasso regression is frequently used for feature selection because it implicitly produces a sparse solution, i.e., only a small portion of the regression coefficients will be non-zero. Using L1-regularization, we can rewrite the least squares problem as:

$$\min_{\beta_0, \beta_1, \dots, \beta_K} \frac{1}{n} \sum_{i=1}^n (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_K x_{iK} - y_i)^2 + \lambda \sum_{k=1}^K |\beta_k| \quad (1)$$

- **L2-regularization** ($p = 2$): Also called **ridge** regression. Ridge regression is frequently used to account for highly correlated feature sets (in an automated way) because it produces a unique solution. Using L2-regularization, we can rewrite the least squares problem as:

$$\min_{\beta_0, \beta_1, \dots, \beta_K} \frac{1}{n} \sum_{i=1}^n (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_K x_{iK} - y_i)^2 + \lambda \sum_{k=1}^K |\beta_k|^2 \quad (2)$$

- **Elastic net regularization**: A linear combination of lasso and ridge regression. Using elastic-net, we can rewrite the least squares problem as:

$$\min_{\beta_0, \beta_1, \dots, \beta_K} \frac{1}{n} \sum_{i=1}^n (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_K x_{iK} - y_i)^2 + \lambda_1 \sum_{k=1}^K |\beta_k| + \lambda_2 \sum_{k=1}^K |\beta_k|^2 \quad (3)$$

All methods reduce coefficient size and help prevent over-fitting. In comparison, L2-regularization is more computationally efficient than L1-regularization, but does not produce sparse solutions and therefore cannot be used for feature selection. Regularization can be applied to many models including linear regression, logistic regression, neural networks, support vector machines, etc.

Training and testing

For prediction problems, we want our model to have high out-of-sample model performance, otherwise known as *generalizability*. To assess out-of-sample model performance, we often partition our data into two sets:

- **Training set**: comprises the observations that are used to *train or fit the model*, i.e., learn the relationship between the features and the target.
- **Testing set**: put aside during the model fitting process and used later to evaluate the performance of the final model.

If we do not use a testing set and simply evaluate the model performance on the training set, this can lead to overfitting. In some cases, we may want to partition our data into three sets: training, testing, and validation:

- **Validation set:** used to choose or tune the hyperparameters.
 - For example, we can use a validation set to determine the optimal value of λ when using regularization. We accomplish this by training the model on the training set with a specified λ . We then evaluate the model performance on the validation set. We repeat this process for many values of λ and choose the λ corresponding to the best model performance. Once our hyperparameter(s) are chosen, we apply the final model to the testing set to evaluate the true out-of-sample performance.

There are many ways to partition the data into training and testing sets. Cross validation is typically used to reduce the variance in the estimates (see the next section). If the data set is sufficiently large, cross validation may not be computationally feasible and a single partition is necessary - typically we use between 70 – 90% of the data for training and validation, and 10 – 30% of the data for testing.

Cross validation

In some prediction-focused applications, especially those with small data sets, different random training and testing set partitions can yield significantly different estimations of model accuracy. Cross validation is a popular approach that can be used to reduce the variance and obtain more stable estimates of accuracy.

- **K-fold cross validation:** the data is partitioned into K equally sized samples. A single sample is used for testing and the remaining $K-1$ samples are used for training. This process is repeated K times, with each of the K samples used exactly once as the test set. The K results can then be averaged to produce a single estimate or displayed graphically (e.g., in a box plot). Although 10-fold cross validation is commonly used, any value of K can be chosen.
- **Repeated K-fold cross validation:** repeat the process of K -fold cross validation many times using different initial partitions of the data. Often considered the gold standard for model evaluation because it reduces both the bias and the variance of the estimations. However, depending on the application, it may require significant computational resources.
- **Stratified K-fold cross validation:** the data is partitioned so that each fold has roughly the same average target value or the same proportion of target classes. In the binary case, this means that each fold has roughly the same number of 1s and 0s. This is commonly used for classification problems where the data set is *imbalanced*, meaning that the ratio of one class to the other is greater than 10 to 1 (e.g., 95 ones and 5 zeros).

Methods for imbalanced data

Imbalanced data is a common pitfall for classification problems and occurs when there are many more observations in one class as compared to the other class. Imbalanced data

is typically a problem when the class ratio is larger than 100:1, but can sometimes be an issue for ratios larger than 10:1. Class imbalance is a problem because a classifier can choose to predict only a single class and achieve accuracy above 99% (or 90% in the 10:1 case). There are many ways to deal with class imbalance, and some popular approaches include:

- *Class weighting*: gives more weight to the smaller class. In other words, there is a larger penalty for misclassifying the smaller class observations.
- *Oversampling*: generate extra observations for the smaller class to balance the data. New observations can be generated randomly or using specialized approaches (e.g., SMOTE).
- *Undersampling*: remove sample observations from the larger class to balance the data. Observations can be removed randomly or using specialized approaches (e.g., Near miss).

For more details, please see Learning From Imbalanced Data by He and Garcia (2009) in the literature folder on the course website.

Metrics for model performance

We will now define some of the most commonly used metrics for evaluating model performance. Different metrics are used for regression and classification problems.

Regression problems

- *Mean squared error (MSE)*: the most commonly used performance measure for regression. Root mean squared error (RMSE) provides context-specific error, meaning that the error is given in the units of interest. Intuitively, RMSE tell us how many units (e.g. dollars) we are off by on average. We can define MSE as follows:

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

- *R-squared (R^2)*: a commonly used unit-less performance measure. R^2 provides a relative performance metric by comparing our model to a *baseline* model that does not include any features. In other words, the baseline model predicts the same outcome for each observation (equal to the average of all training set target values). Intuitively, R^2 is the percentage of target variability that can be explained by the model. We can formally define R^2 as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2},$$

where $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$ is the mean of the true targets. R^2 typically ranges from 0 to 1, where a value of 1 indicates a perfect fit.

Classification problems

For most classification problems, the model outputs a probability of each class, not the class itself. Suppose we want to predict whether or not a person has a disease. This is a binary problem and our model will output the probability of disease. We then choose a *threshold rule* (sometimes called a *rounding threshold*) such that people with probabilities above the threshold are classified as having the disease and people below are not. We can choose the threshold value just like any other hyperparameter. Before discussing performance measures, we need to define the following:

- *True positive rate (TPR)*: $\frac{TP}{TP+FN}$. Measures the proportion of positives ($y = 1$) that are correctly identified. Also known as *sensitivity*.
- *True negative rate (TNR)*: $\frac{TN}{TN+FP}$. Measures the proportion of negatives ($y = 0$) that are correctly identified. Also known as *specificity*.
- *False positive rate (FPR)*: $\frac{FP}{TN+FP}$. Measures the proportion of negatives ($y = 0$) that are incorrectly identified. Also known as *Type I error*.
- *False negative rate (FNR)*: $\frac{FN}{TP+FN}$. Measures the proportion of positives ($y = 1$) that are incorrectly identified. Also known as *Type II error*.

Note that $TPR + FNR = 1$ and $TNR + FPR = 1$. Also note that error is highly dependent on threshold value selection (e.g. a lower threshold will increase the number of false positives). Depending on the problem, there may be a certain preference for one type of classification error over another (e.g. higher false positives are preferred in disease prediction). Now some performance measures:

- *Classification accuracy*: $\frac{TP+TN}{TP+TN+FP+FN}$. Measures the percentage of targets correctly classified. Suppose we have 100 binary targets (some 1s and some 0s). If we predict 95 correctly, then our classification accuracy is 0.95 or 95%.
- *Confusion matrix*: a visualization of the true positives, true negatives, false positives, and false negatives. See Figure 3 for an example based on Lecture 3 (predicting heart disease).

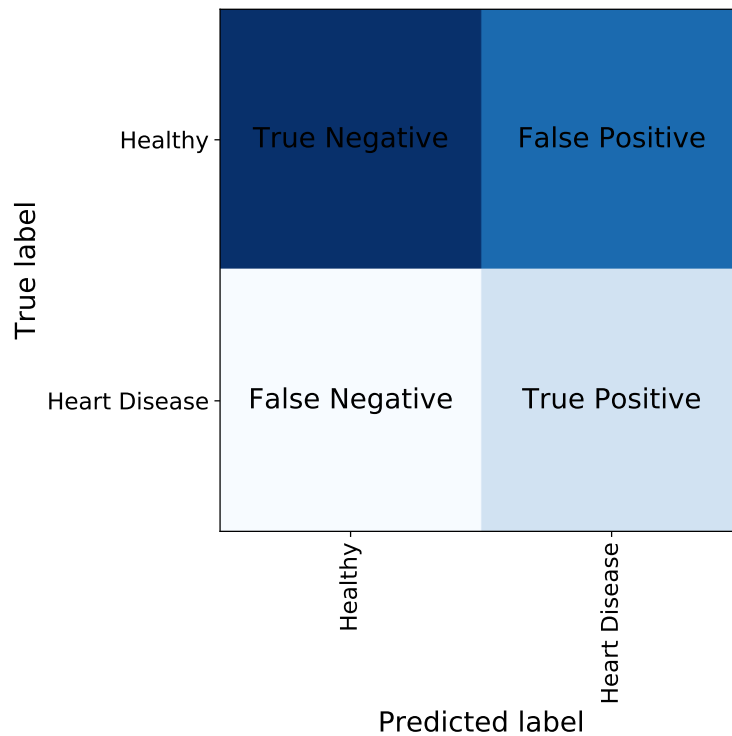


Figure 3: Example confusion matrix for a binary problem. Note that Heart Disease is equal to 1 (and healthy is 0).

- *Receiver operating curve (ROC)*: visualizes the trade-off between the TPR and FPR. In particular, the ROC illustrates the TPR and FPR as the discriminant threshold is varied. See Figure 4. The performance of a *random classifier* (i.e., guessing) is given by the red dotted line. Note the direction of classifier improvement.

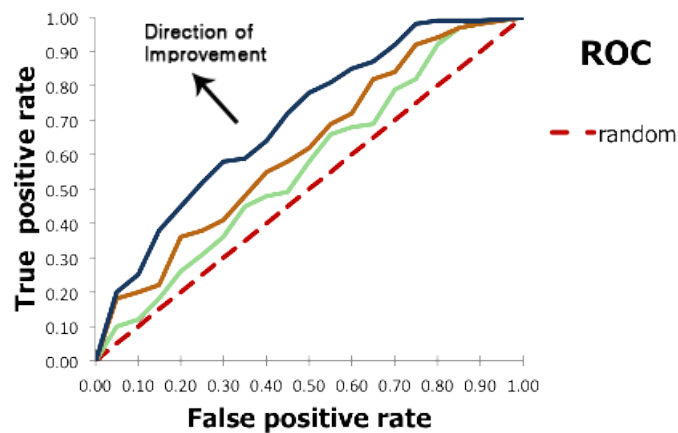


Figure 4: Receiver operating curve for a binary problem. Each curve corresponds to a different model.

- *Area under the ROC curve (AUC)*: the most commonly used performance metric for classification problems. AUC measures the area under the ROC curve. An AUC of 1 indicates a perfect classifier, while an AUC of 0.5 indicates random guessing. AUCs below 0.5 imply that the model is worse than guessing. AUC is commonly used because it is a single metric that accounts for the trade-off between true positives and false positives.