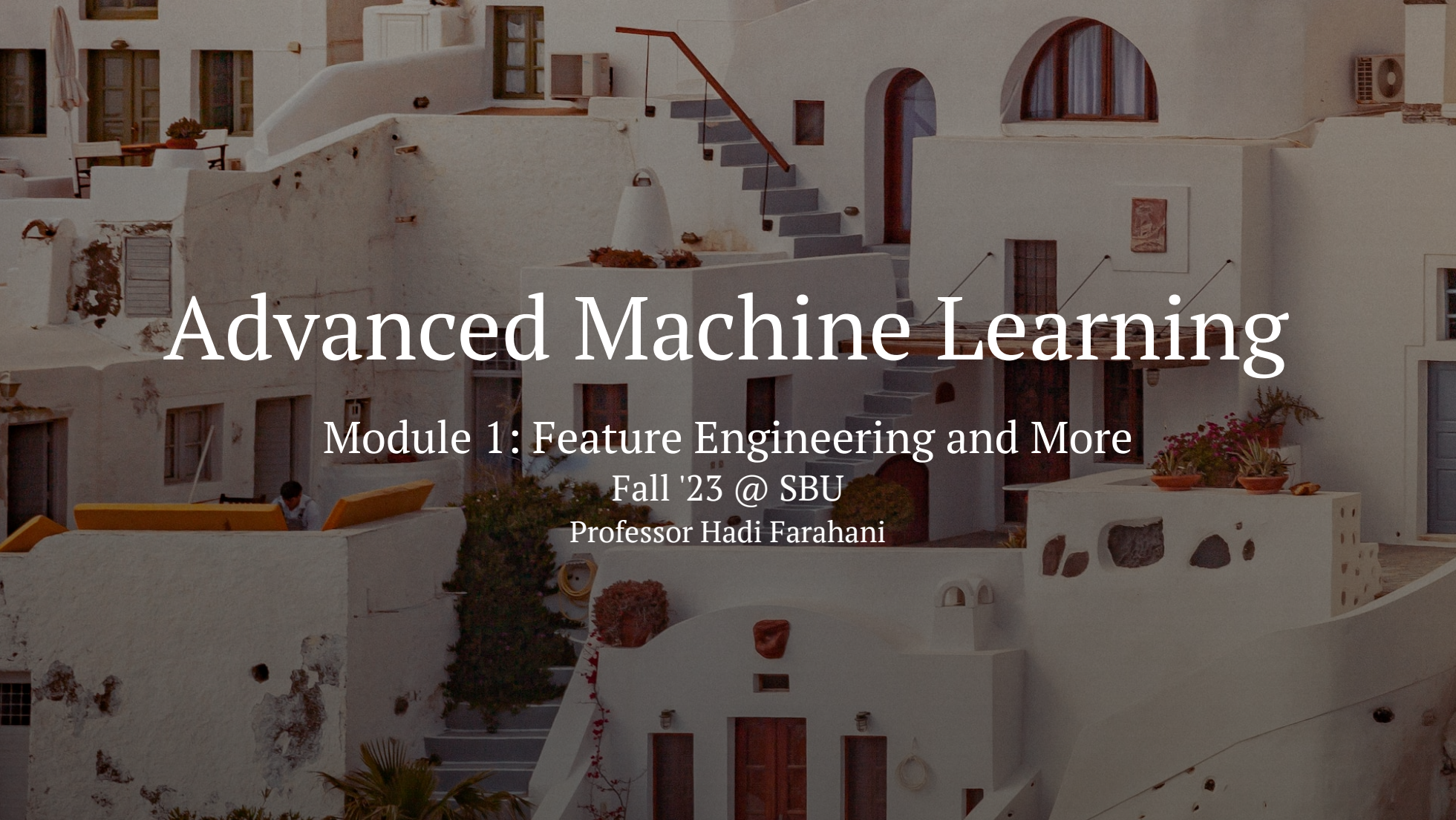


Advanced Machine Learning

Module 1: Feature Engineering and More

Fall '23 @ SBU

Professor Hadi Farahani

Module Overview

In this module, we'll be covering the following topics:

- Feature engineering and what it is
- **Level 0 feature engineering**: Handling null values, scaling, normalization, handling strings
- **Level 1 feature engineering**: Feature engineering for model interpretability, capturing dependencies, outlier detection, unbalanced datasets
- **Level 2 feature engineering**: Timeseries, textual feature engineering (words), unstructured data and more
- *Bonus*: Automated feature engineering

Introduction to Feature Engineering

So, what's this feature engineering I keep hearing about?

According to Wikipedia:

> Feature engineering or feature extraction or feature discovery is the process of extracting features (characteristics, properties, attributes) from raw data.

Great, but what does that *really* mean?

In short, it means a bunch of different things – anything from data cleaning to outlier detection to dependency capturing falls under the umbrella of feature engineering.

In short, it's how we process our data to make our machine learning model work better.

Prelude: We Need To Be Careful

When we're training a machine learning model, we're looking to extract as much information as we can from every single feature.

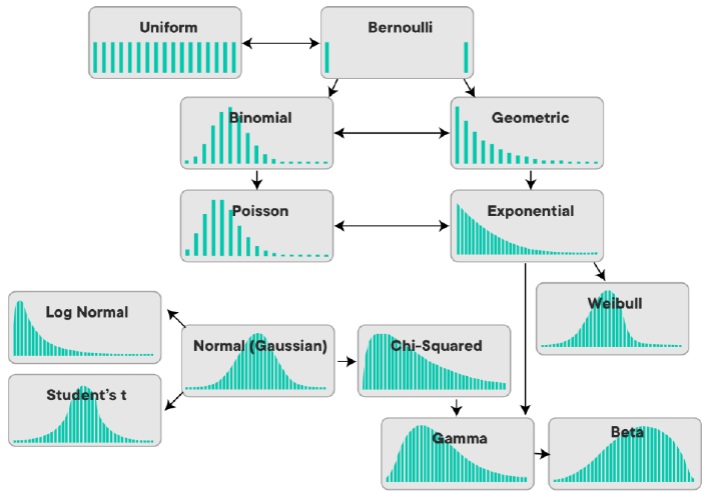
- Everything from the range of values, to distribution, etc. may contain valuable information that can be used to better our predictor and we need to take care to not inadvertently lose this information. We need to be sure of our transformations, and that nothing of value is being lost.

One of the most important factors is maintaining feature distribution.

Prelude: We Need To Be Careful

But why is it so important?

- Preserving the distribution becomes crucial when building statistical models that assume certain distributional properties of the variables.
- In some cases, maintaining the distribution helps preserve the interpretability of the model.
- Altering the distribution might introduce instabilities or result in models that are sensitive to small changes in the input. Retaining the original distribution can help produce consistent and stable model behavior.
- Altering the distribution of a feature can introduce bias into the data. For example, in skewed distributions, the majority class may dominate the modeling process, leading to biased decision-making.



If we're using non-parametric models that do not make assumptions about the underlying data, such as Random Forest, we no longer need to worry about distribution as *much*.

So in summary, when applying transformations to our data, we need to worry about preserving as much information as possible.

Level 1: Basic Feature Engineering

Handling Missing Values

Various strategies for handling missing data, including imputation techniques (e.g., mean, median, mode imputation, regression imputation) and missingness indicators like missingness patterns.

Student			
Enroll	S_name	Address	Contact
1001	Rahul	Gwalior	NULL
1002	Rakesh	NULL	9977862211
1003	bharta	gwalior	NULL
1004	atulia	bihar	465748957

> The act of replacing missing data with statistical estimates of missing values is called imputation.

- *Given our task*, we need to be careful as to how we handle null values.

There are several methods for handling null data, and we need to pick the best method given our data, and what we're looking to achieve.

Method 1: Dropping All Null Values

Perhaps counterintuitively, this method is actually a very good method for imputing data points.

Complete Case Analysis (CCA), also called list-wise deletion of cases, consists of discarding those observations where the values in any of the variables are missing.

If our dataset allows this (i.e. our dataset doesn't shrink too much) after dropping all missing columns, CCA is worth considering.

```
import pandas as pd

df = pd.read_csv(filepath)

df = df.dropna()
```

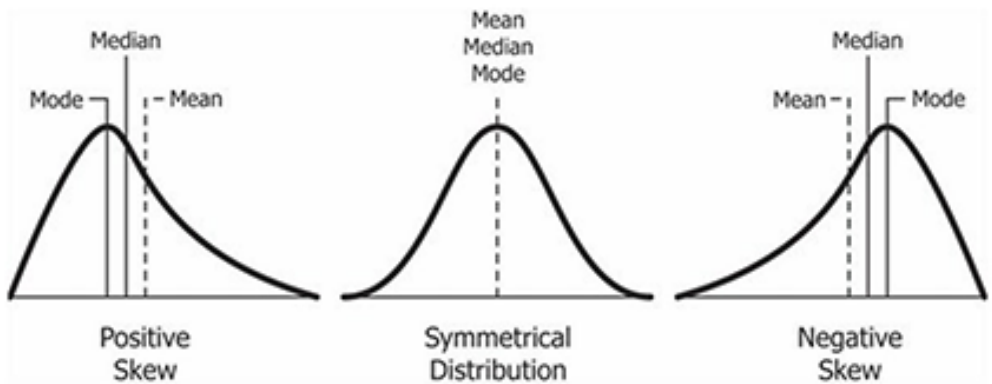
(We did this last time)

Method 2: Performing Mean or Median Imputation

Mean or median imputation consists of replacing missing values with the variable mean or median. This can only be performed in numerical variables. The mean or the median is calculated using a train set, and these values are used to impute missing data in train and test sets, as well as in future data we intend to score with the machine learning model.

Use mean imputation if variables are normally distributed and median imputation otherwise.

- Mean and median imputation may distort the distribution of the original variables if there is a high percentage of missing data.



Method 2: Performing Mean or Median Imputation

While we could implement everything by hand, a potential solution for this would be to use Scikit Learn's SimpleImputer.

With SimpleImputer, we can pick a strategy and impute missing values based on that strategy.

```
from sklearn.impute import SimpleImputer

imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')

imp_mean.fit([[7, 2, 3], [4, np.nan, 6], [10, 5, 9]])

X = [[np.nan, 2, 3], [4, np.nan, 6], [10, np.nan, 9]]

imp_mean.transform(X) # <-- Final results
```

From documentation

> Onto code!

Handling Missing Categorical Features

We can also replace missing strings in categorical values with “missing” – almost as if it were a category of its own.

- Equivalent to using the constant strategy when using SimpleImputer

However, this is obviously very dangerous, and you should absolutely test it out.

0 as Feature Value

However, not all missing values might look like `NaN`.

Sometimes, we're faced with features with that can have a value of 0, effectively making the model act like some of these values are missing

> Why is this bad?

0 as Feature Value

Having 0 as a potential value for a feature in a dataset may not necessarily be bad, and it depends on the context and the specific attribute being represented.

- However, in some cases, having 0 as a potential value can lead to issues during training and inference in machine learning models.

0 as Feature Value

Some issues include:

- Bias and interpretation: A feature value of 0 can introduce bias and affect the interpretation of the model. If a feature is expected to have a non-zero value but it is encoded as 0, the model may incorrectly learn that the absence of that feature significantly affects the model's understanding, and by proxy, its prediction.
- Issues during scaling: Many machine learning algorithms, such as neural networks, depend on appropriate feature scaling or normalization. When a feature has 0 as a potential value, it can hinder these techniques, leading to imbalanced gradients or difficulties in convergence.

However, it is important to note that the impact of including 0 as a potential value depends on the specific task, data distribution, and the algorithms and techniques being used. In some cases, it might not cause any issues. It is indeed a domain-specific consideration to decide if including 0 in the feature space is appropriate or if it needs preprocessing, feature engineering, or normalization techniques to handle it effectively.

Method 3: Imputation Through Other Machine Learning Models

Sometimes, we want a more dynamic method for filling in gaps in our data and sometimes, we want to simply remove 0 values in our data.

A way to do this is imputation via other machine learning models.

Simply put, we train a model (usually something ‘light’ and relatively computationally inexpensive) on other features, and after training, we predict new values for this specific column. Then, we train our final model on all the features.

Your Turn!

Method 3: Imputation Through Other Machine Learning Models

There are a number of advantages to this specific method:

1. **Utilize relationships between variables**: Machine learning models can capture complex relationships between variables. By training a model on other features, you can leverage these relationships to predict missing values more accurately.

2. **Handles non-linear relationships**: Machine learning models can handle non-linear relationships, unlike traditional imputation methods (e.g., mean imputation or regression imputation). This makes them more capable of imputing missing values accurately in non-linear scenarios.

3. **Minimize bias in imputation**: By using various features to predict missing values, machine learning models reduce the potential bias that may arise when imputing based on a single feature or summary statistics.

4. **Improve downstream analysis**: Imputing missing values accurately helps to retain more data for analysis. This can lead to improved performance and insights gained from subsequent machine learning or statistical models trained on the imputed dataset.

> Using mean or median imputation is a much simpler version of this method.

But what do we do when there are multiple columns with missing values?

Method 4: Multivariate Imputation

Instead of using only one variable to estimate missing values, multivariate imputation methods utilize the entire set of variables.

This means that the missing values of a variable are predicted based on the other variables in the dataset.

- One technique for multivariate imputation is called **Multivariate imputation by Chained Equations (MICE)**.

MICE

MICE involves several steps:

1. Initially, a simple univariate imputation, such as median imputation, is performed for each variable with missing data.
2. Then, a specific variable, let's say `var_1`, is chosen, and it's missing values are reset to missing.
3. A model is created to predict `var_1` using the remaining variables in the dataset.
4. The missing values of `var_1` are replaced with the new estimates obtained from the model.
5. Steps 2 to 4 are repeated for each of the remaining variables.

This cycle of imputation is performed multiple times, typically around 10 times, retaining the imputation values after each round. The aim is for the distribution of the imputation parameters to converge by the end of the cycles.

Various approaches can be used to model each variable with missing data based on the other variables. Examples include linear regression, Bayes, decision trees, k-nearest neighbors, and random forests.

MICE

This cycle of imputation is performed multiple times, typically around 10 times, retaining the imputation values after each round. The aim is for the distribution of the imputation parameters to converge by the end of the cycles.

Various approaches can be used to model each variable with missing data based on the other variables. Examples include linear regression, Bayes, decision trees, k-nearest neighbors, and random forests.

Extra: Adding a Missing Value Indicator

A missing indicator is a binary variable that specifies whether a value was missing for an observation (1) or not (0).

This missing value indicator can serve as an additional feature, capturing the missingness pattern.

It allows the model to learn the relationship between missing values and the target variable, potentially leading to improved predictive performance.

SimpleImputer also allows us to add a Missing Indicators column if so needed.

> Try it out yourself!!

Handling Categorical Variables

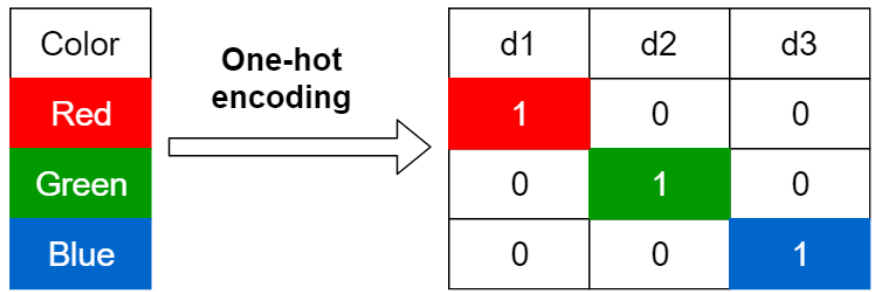
Advanced techniques for encoding categorical variables, including one-hot encoding, label encoding, target encoding, and hashing trick, and handling high-cardinality categorical features.

Here, we'll talk about the pros and cons of different methods for encoding categorical values.

Method 1: One-hot Encoding

One-hot encoding is a technique used to convert categorical data into a binary format that can be easily understood by machine learning models.

In this technique, each category is represented by a binary vector where only one bit is "hot" (1) and the rest are "cold" (0).



- This means that each category is converted into a unique binary vector, and the value of the "hot" bit represents the presence or absence of that category.

Method 1: One-hot Encoding

There are several reasons why might want to use this method:

- It's easy to implement
- Handles non-ordinal categories, and does not assume any order in the labels, preventing false predictions

But we need to take care, because this method has several drawbacks:

- Increases drawbacks
- Ignores potential similarities, as one-hot encoding treats every category as independent

However, to help alleviate these problems, we can perform one-hot encoding on frequent categories only. The less frequent and rare categories can be grouped together into a single, unique category.

Method 2: Ordinal Number Encoding

Ordinal encoding consists of replacing the categories with digits from 1 to k (or 0 to k-1, depending on the implementation), where k is the number of distinct categories of the variable.

The numbers are assigned arbitrarily. Ordinal encoding is better suited for nonlinear machine learning models, which can navigate through the arbitrarily assigned digits to try and find patterns that relate to the target.

Much like one-hot encoding, this method comes with pros and cons;

Pros:

- Retains ordinal information
- Good for linear algorithms

Cons:

- Arbitrary category assignment
- Misleading magnitude
- Not suitable for non-linear algorithms

Method 3: Replace Categories with Frequencies

In count or frequency encoding, we replace the categories with the count or the percentage of observations with that category. That is, if 10 out of 100 observations show the category blue for the variable color, we would replace blue with 10 when doing count encoding, or by 0.1 if performing frequency encoding

The assumption is that the number of observations per category is somewhat predictive of the Target. We don't need to use raw counts here, mean (%) can also work well.

However, there is the very obvious drawback that if two categories have the same frequency, they'll have the same label.

We can extend this method and order our categories by frequency. This order can then be used for ordinal encoding.

Feature Scaling and Normalization

Transforming all values in a feature at once is commonly done in feature engineering, whether it be for scaling, normalization or standardization.

> But what's the difference?

Scaling vs. Normalization vs. Standardization

- Scaling simply changes the range of values in a feature, and does not interfere with distribution.
- Normalization rescales the values into a range of [0,1]. This might be useful in some cases where all parameters need to have the same positive scale. If the distribution is not Gaussian or the standard deviation is very small, the min-max scaler works better.

person_name	Salary	Year_of_ experience	Expected Position Level
Aman	100000	10	2
Abhinav	78000	7	4
Ashutosh	32000	5	8
Dishi	55000	6	7
Abhishek	92000	8	3
Avantika	120000	15	1
Ayushi	65750	7	5

The attributes salary and year_of_experience are on different scale and hence attribute salary can take high priority over attribute year_of_experience in the model.

- Standardization rescales data to have a mean (μ) of 0 and standard deviation (δ) of 1 (unit variance). It is useful when the feature distribution is Normal or Gaussian.

But Why is Normalization Important in the First Place?

Normalization is important because it brings all feature values to a similar scale, making the features comparable and preventing models from being dominated by certain features.

Here are a few reasons why normalization is done even when feature distribution is important:

1. **Prevents bias**: Normalization removes any inherent bias that may occur due to the scale of the features. If features have different scales, the model might give more importance to features with larger values, even if they may not be as important. Normalizing features eliminates this bias and ensures fair consideration for all features.

2. **Improves convergence**: Some machine learning algorithms, such as gradient descent, converge faster when the features are normalized. This is because normalization reduces the range of feature values and helps the algorithm find the optimal solution more quickly.

But Why is Normalization Important in the First Place?

3. **Regularization effects**: Many regularization techniques, like L1 or L2 regularization, assume that features are on a similar scale. If features are not normalized, regularization might penalize some features disproportionately, resulting in an undesired impact on the model's performance.

4. **Comparison across models**: Normalizing features allows for fair comparison across different models or algorithms. If different models are trained on features with different scales, it becomes difficult to determine which model is performing better.

Overall, while feature distribution is indeed important, normalization complements this aspect to ensure that features are appropriately scaled for optimal performance and comparability.

Method 1: Logarithmic Scaling

The logarithm function is commonly used to transform variables. Not only is it simple and easy to implement, it can help in a number of different ways:

- Great at handling skewed data
- Reduces the impact of extreme values
- The logarithm function is commonly used to transform variables. It has a strong effect on the shape of the variable distribution and can only be applied to positive variables.
- Converting multiplicative relationships into additive ones

However, it can only be applied to positive variables and may not be the best option when we want to compare absolute values as opposed to relative values.

> Using Pandas' `map` or `apply`, you should try doing this, then testing to see if it's had an effect on our model performance

> When we have negative values, we can use the reciprocal function (1/x), but using this function requires us to have non-zero values for every row.

Method 2: Min-Max Scaling

Minimum and maximum value from data is fetched and each value is replaced according to the following formula;

$$v' = \frac{v - min(A)}{max(A) - min(A)}(new\_max(A) - new\_min(A)) + new\_min(A)$$

<br / >

Method 3: Z-Score Normalization

Z-Score normalization scales all values between 0 and 1 and has the following formula:

$$New\_Value = \frac{x - \mu}{\delta}$$

Where x is the original value, μ is the mean of the data, and δ is the standard deviation of the data.

When to Use What Method

The methods shown in the previous slide are by no means exhaustive; these only serve as a reminder and/or introduction to various commonly used methods in feature engineering.

There's a wide variety of different methods to use for scaling, normalization and standardization. What's important is to first figure out which of these three you're looking to do.

> Further reading is Chapter 4 and 8 in the Feature Engineering Cookbook

Variable Discretization

Discretization is the process through which we can transform continuous variables, models or functions into a discrete form. These discrete values are then handled as categorical values.

It works, because:

- It helps the model better identify patterns
- Discretization can help prevent overfitting by reducing noise and capturing the underlying trend or pattern in the data.
- Discretization can help handle outliers and missing values in a more robust manner.
- By discretizing variables, the model becomes less reliant on the specific distribution of the data.

The main challenge in discretization is identifying the thresholds or limits to define the intervals into which the continuous values will be sorted.

Method 1: Dividing Variables into Equal Length Intervals

The process of equal-width discretization involves sorting variable values into intervals that have the same width.

The number of intervals is chosen without specific criteria, while the width is determined based on the range of variable values and the desired number of bins.

For the variable X, the width of each interval is calculated as follows:

$$Width = \frac{Max(X) - Min(X)}{Bins}$$

> When is this method a good idea? When is it not?

Method 2: Dividing Variables in Intervals of Equal Frequency

Equal-frequency discretization is a method that categorizes the values of a variable into intervals that contain an equal proportion of observations.

The width of each interval is determined by quantiles, which means that the intervals may have different widths. To implement this technique, the continuous variable is divided into N quantiles, where N is determined by the user.

- This discretization technique is especially beneficial for skewed variables as it evenly distributes the observations among the different bins.

Method 3: Performing Discretization with K-Means Clustering

The k-means algorithm is used in discretization to identify clusters, with each cluster representing an interval. The number of clusters (k) is determined by the user.

The k-means clustering algorithm consists of two main steps.

1. In the initialization step, k observations are randomly selected as the initial centers of the k clusters, and the remaining data points are assigned to the nearest cluster.
2. In the iteration step, the cluster centers are recalculated as the average of all observations within each cluster, and the observations are reassigned to the closest newly created cluster.

This iteration step continues until the optimal k centers are obtained.

> See Scikit-Learn's KBinsDiscretizer

Alternatively, we could use decision trees for discretization.

Evaluation Metrics for Feature Engineering

After tuning and tweaking our data, it's good to have an idea of how to evaluate our final machine learning model.

On top of that, picking the right metric to train our model with is also incredibly important.

> Why is the training metric so important?

Here, we'll briefly go over a few popular functions for regression and classification tasks, and when to use which.

Regression Metrics

Mean Squared Errors (MSE)

MSE is a good tool for comparing regression models applied to the same problem. It's calculated as follows:

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

But unfortunately, it's a bit hard to tell just how good your model is doing. Alternatives include:

- **Root Mean Squared Error (RMSE)**: Calculated by simply taking the root of MSE. The result will resemble the original scale of target, thus giving us a better view of how good our model(s) is (are) doing.
- **R squared**: Perhaps even better, as it perfectly correlates with MSE and its values range between 0 and 1, making comparisons even easier. This metric is defined as follows:

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

Regression Metrics

Root Mean Squared Log Error (RMSLE)

Another common transformation of MSE, it uses the formula below:

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(log(\hat{y}_i + 1) - log(y_i + 1))^2}$$

Because of the logarithm applied to predictions and ground truth before the other operations (such as squaring, averaging, etc.), you're not penalizing huge differences between the predicted and actual values, especially when both are large numbers.

In other words, you use RMSLE when the scale of your predictions with respect to the scale of the ground truth is what we care about.

Regression Metrics

Mean Absolute Error (MAE)

Simply put, the absolute difference between predictions and targets, as given below:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

MAE is not sensitive to outliers, so it might be best to use MAE over MSE when there are many outliers in our training data.

For more information, see this link.

Classification Metrics

Accuracy

The most common metric used in classification tasks is accuracy, and can be obtained by calculating the ratio between the correct predictions and the number of answers:

$$Accuracy = \frac{correct\_answers}{total\_answers}$$

This metric is great for real-world scenarios where we're only looking to evaluate model performance. However, when dealing with class imbalance, this metric is no longer as useful.

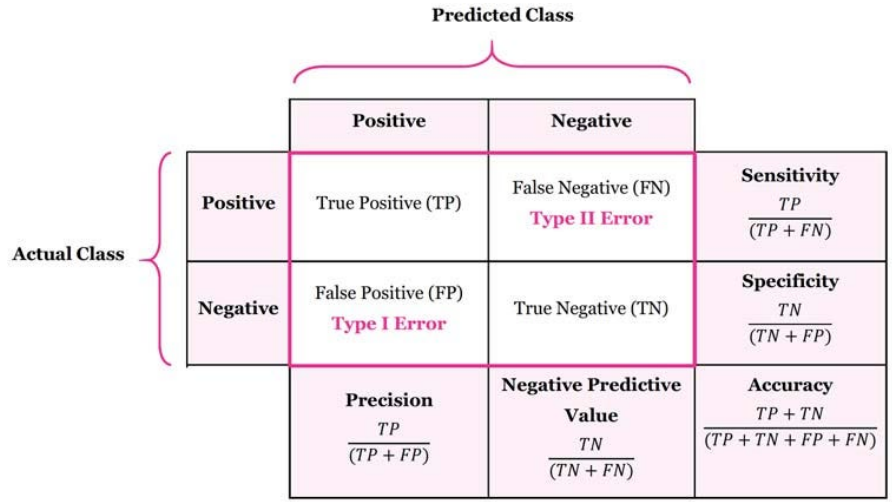
Classification Metrics

Precision and Recall

Precision (also known as specificity) is basically the accuracy of positive cases:

$$Precision = \frac{TP}{TP + FP}$$

Using this metric forces your model to only predict positive for examples it has high confidence in, and is, in fact, the intended purpose of the measure.



However, if you're looking to predict as many positives as possible, it would be best to calculate recall as well:

$$Recall = \frac{TP}{TP + FN}$$

(Also known as coverage, sensitivity or even true positive rate)

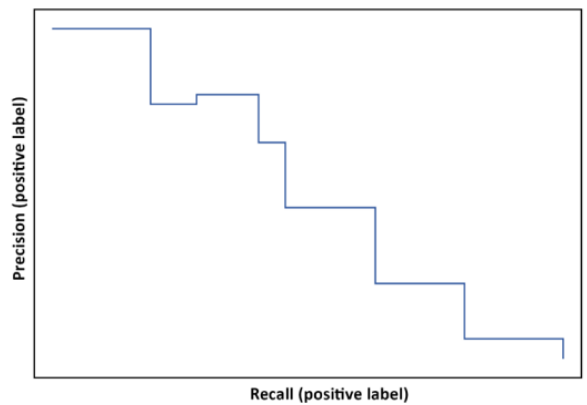
Classification Metrics

Precision and Recall

Striking a balance between precision and trade-off is called the precision/recall trade-off. There's the **F1 score**, that takes both precision and recall into account, and is given by the harmonic mean of precision and recall, and is commonly considered the best solution.

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

If we have a high F1 score, it means our model has high precision, or high recall, or both.



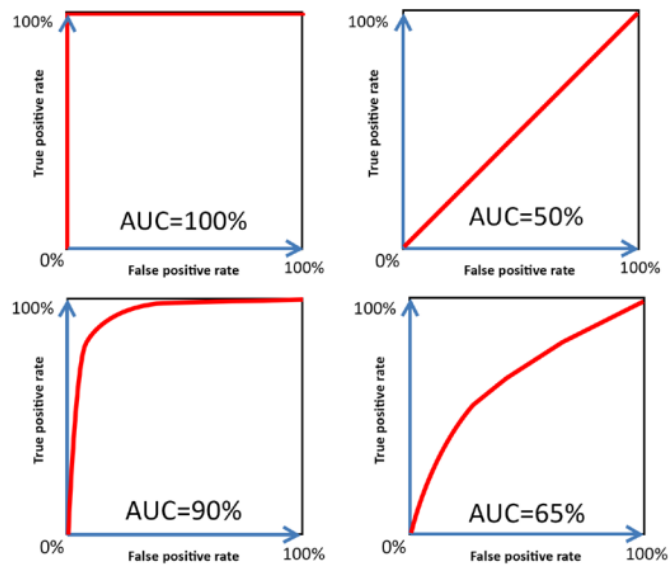
Classification Metrics

ROC Curve

The ROC curve, also known as the receiver operating characteristic curve, is a visual representation that assesses the effectiveness of a binary classifier and allows for the comparison of multiple classifiers.

It serves as the foundation for the ROC-AUC metric, as the metric represents the area under the ROC curve. The ROC curve depicts the true positive rate (recall) plotted against the false positive rate (the proportion of negative instances incorrectly classified as positive). In essence, it is the complement of the true negative rate (the proportion of negative examples correctly classified).

Ideally, a ROC curve of a well-performing classifier should quickly climb up the true positive rate (recall) at low values of the false positive rate. A ROC-AUC between 0.9 to 1.0 is considered very good.



Level 2: Slightly Less Basic Feature Engineering

Outlier Detection

An outlier is a data point that is significantly different from the remaining data. These outliers might contain valuable information about our data as a whole, or they might not be nothing more than a statistical anomaly.

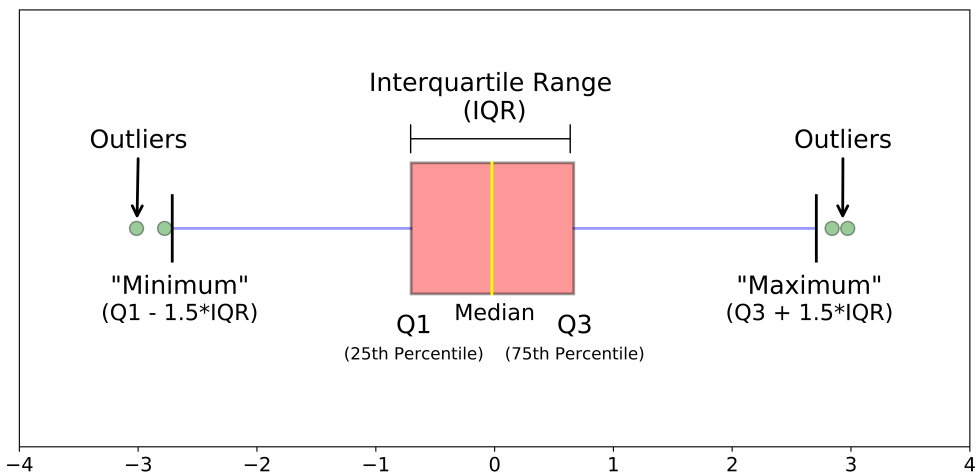
Outliers, when they're not important, are a source of noise in our model, and removing them can help improve our accuracy, as keeping them inside our training data can potentially "confuse" our model.

Method 1: Box-Plot

One popular method is applying the inter-quantile range (IQR) proximity rule, also known as the Box-Plot Method where we define a range for our data, and should a point fall outside this range, it is considered an outlier.

Upper boundary = 75th quantile + (IQR * 1.5) Lower boundary = 25th quantile - (IQR * 1.5) IQR = 75th quantile - 25th quantile

> However, it is also common practice to find extreme values by multiplying the IQR by 3.



Method 2: Z-Score for Outlier Detection

Another commonly used method is using Z-Scores for outlier detection and removal.

Z-scores are a statistical tool used for outlier detection by measuring how far away a data point is from the mean in terms of standard deviations.

- This method is preferable in situations where the distribution of the data is approximately normal or symmetric.

Z-scores help to identify extreme values that deviate significantly from the mean, allowing analysts to determine whether these values are outliers. The Z-Score is calculating through the following formula:

$$Z = \frac{x - \mu}{\sigma}$$

Where μ is the mean, and σ is the standard deviation.

Method 2: Z-Score for Outlier Detection

By setting a threshold, usually around 2 or 3 standard deviations, observations beyond this limit can be flagged as potential outliers.

This method is particularly useful when the data does not exhibit strong skewness or extreme outliers, making it applicable in various fields such as finance, quality control, and identifying abnormal behavior in statistical analysis.

But, if the data is skewed, we could use modified z-scores instead. Instead of relying on the mean and standard deviation, modified z-scores use the median and median absolute deviation (MAD) to identify outliers.

Method 2.5: Modified Z-Score

Modified Z-Score is given through the following formula:

$$ModZScore = 0.6745(x_i - \tilde{x})/MAD$$

Where $\tilde{x}$ is the median of the dataset, and MAD is the median absolute deviation of the dataset.

Modified z-scores are more robust against outliers and provide a more accurate representation of the central tendency and dispersion of the data, however, when our data is symmetrical and normally distributed, it might be best to use regular z-score instead.

Other Methods

A couple other notable methods include:

- Local Outlier Factor (LOF) method: LOF identifies outliers by comparing the local density of a data point with its neighbors. Points with significantly lower density are considered outliers. How to implement
- SVM can be used to detect outliers by identifying data points that are difficult to be classified or far from the decision boundary.

> So in general, with outlier detection the key is *how* the outliers are detected. This is one of those scenarios where knowing how our data is distributed is incredibly important.

Feature Selection

Feature selection is the process of selecting a subset of relevant features from a larger set of features to enhance model performance.

Feature selection is crucial in machine learning as it helps to identify the most relevant and influential features in a dataset, thereby improving model accuracy, reducing computational costs, and eliminating noise and irrelevant data.

- It helps to create simpler and interpretable models, enhance generalization, and reduce overfitting by focusing on the most informative features.

Feature Selection Methods

There are plenty of methods we can use for feature selection, and like everything talked about so far in this module, it depends on what works best for your data. For example:

- Removing variables with low variance
- Conducting hypothesis tests to pick features
- Removing features one by one to see how it affects our score.
- etc.

Feature Interaction and Polynomial Features

Exploring techniques like interaction terms, polynomial features, and different feature engineering frameworks (e.g., Featuretools) to capture non-linear relationships and complex interactions between features.

Polynomial features work because they help capture non-linear relationships between the features and the target variable in a dataset. Linear regression models only account for linear relationships, whereas polynomial features allow for more flexible and complex relationships.

When we add polynomial features to the dataset, we introduce interactions between the features by including higher-order terms. For example, if we have a feature x, adding a quadratic term x^2 allows us to capture a non-linear relationship between x and the target variable. By including polynomial terms, the model becomes more expressive and can better fit the data.

Polynomial features also help in situations where it is difficult to uncover the underlying relationship between the features and the target variable using linear models. By allowing for non-linear relationships, polynomial features are able to capture patterns that would otherwise be missed by a linear model.

However, it is important to note that adding polynomial features can also introduce overfitting, where the model becomes overly complex and performs poorly on new, unseen data. Therefore, it is crucial to carefully select and scale the polynomial features to prevent overfitting and achieve better generalization.

Further Feature Engineering Topics

- Feature engineering for model interpretability

> Techniques like feature importance calculation (e.g., using permutation importance, feature contribution, SHAP values) and feature interaction analysis (e.g., partial dependence plots, interaction effects) for interpretability and model understanding purposes.

- Feature Engineering for Imbalanced Datasets Techniques to handle imbalanced datasets, including oversampling (e.g., SMOTE, ADASYN), undersampling, and combining various sampling strategies with the feature engineering process to improve model accuracy.

Level 3: Not-Basic Feature Engineering

Level 3: Not-Basic Feature Engineering

Feature engineering is the process of creating new input features from existing data to better represent patterns and relationships in the data for machine learning models. The approach to feature engineering can differ between tabular and non-tabular data types.

While feature engineering for tabular data often involves manipulating columns and creating new features based on the data, feature engineering for non-tabular data focuses on extracting relevant information from the data and representing them in a format suitable for machine learning models.

Level 3: Not-Basic Feature Engineering

- **Time-Series Feature Engineering**: Techniques for handling time-series data, such as creating lag features, rolling/window statistics, and Fourier/wavelet transformations to capture temporal patterns. (chapter 7 in feature engineering book) Also chapter 10

- **Textual Feature Engineering**: Covering methods like TF-IDF, word embeddings (e.g., Word2Vec, GloVe), and topic modeling (e.g., Latent Dirichlet Allocation) for extracting meaningful features from textual data.

Feature Extraction from Unstructured Data: Techniques for extracting features from unstructured data like images (e.g., using CNN-based architectures) and audio (e.g., MFCC, spectrograms) to enable machine learning models to analyze such data effectively.

- **Feature Engineering for Deep Learning**: Exploring techniques to engineer features for deep learning models like convolutional neural networks (CNN), recurrent neural networks (RNN), and transformers by using techniques like pre-training, transfer learning, and attention mechanisms.