

MACHINE LEARNING

CHAPITRE 2

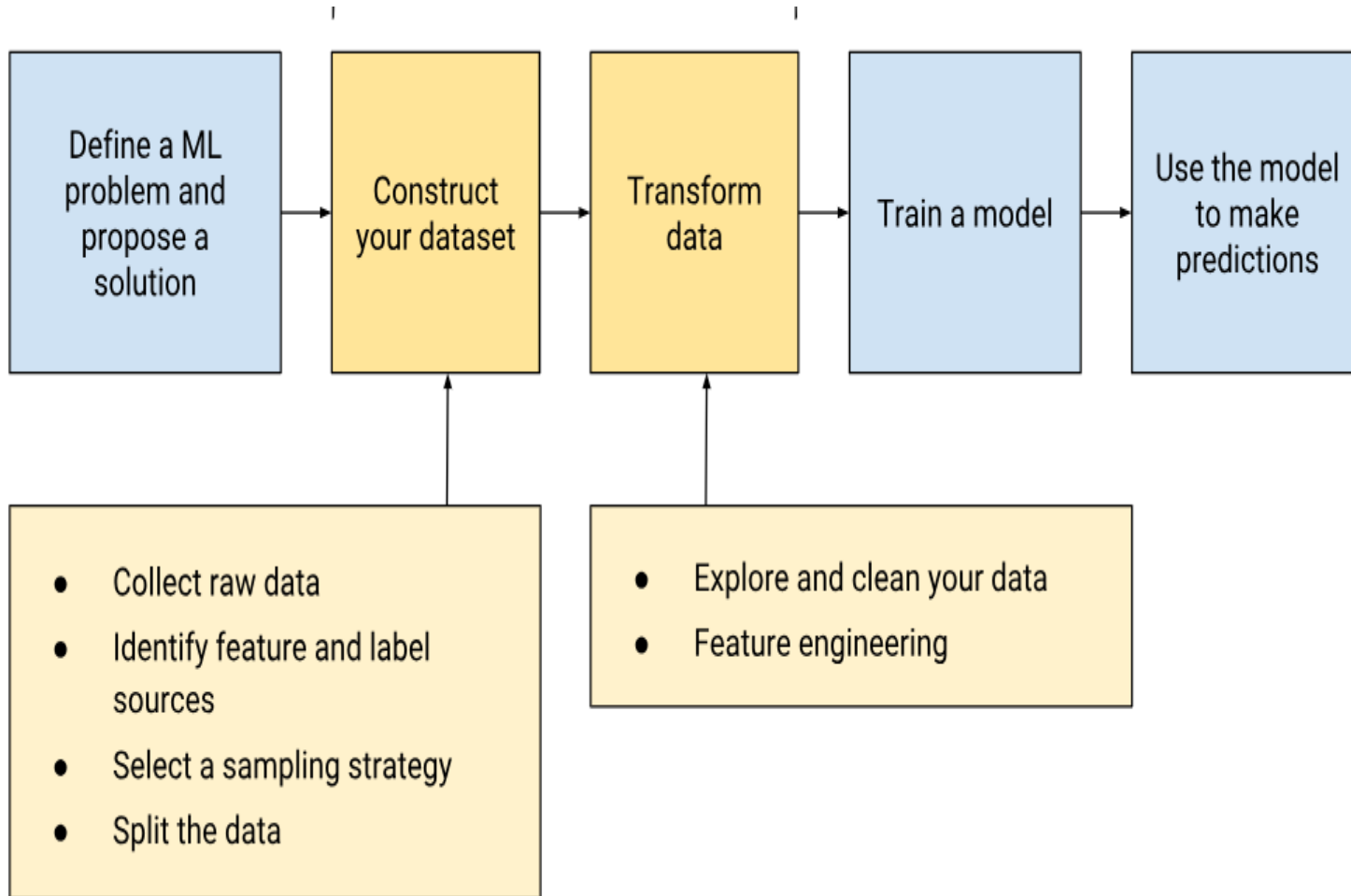
Pr. BENLAHMAR EL HABIB



Data

Data Preparation (Data pre-processing)

The Process for Data Preparation and Feature Engineering



Source : <https://developers.google.com/machine-learning/data-prep>

Steps to Constructing Your Dataset

- To construct your dataset (and before doing data transformation), you should:
 1. Collect the raw data.
 2. Identify feature and label sources.
 3. Select a sampling strategy.
 4. Split the data.
- These steps depend a lot on how you've framed your ML problem. Use the self-check below to refresh your memory about problem framing and to check your assumptions about data collection.

Self-check of Problem Framing and Data Collection Concepts

- You're on a brand new machine learning project, about to select your first features. How many features should you pick?
 - ▣ Pick as many features as you can, so you can start observing which features have the strongest predictive power.

Self-check of Problem Framing and Data Collection Concepts

Pick as many features as you can, so you can start observing which features have the strongest predictive power.

- Start smaller. Every new feature adds a new dimension to your training data set. When the dimensionality increases, the volume of the space increases so fast that the available training data become sparse. The sparser your data, the harder it is for a model to learn the relationship between the features that actually matter and the label. This phenomenon is called "the curse of dimensionality."

Pick 4-6 features that seem to have strong predictive power.

- You might eventually use this many features, but it's still better to start with fewer. Fewer features usually means fewer unnecessary complications.

Pick 1-3 features that seem to have strong predictive power.

- It's best for your data collection pipeline to start with only one or two features. This will help you confirm that the ML model works as intended. Also, when you build a baseline from a couple of features, you'll feel like you're making progress!

Self-check of Problem Framing and Data Collection Concepts

- Your friend Sam is excited about the initial results of his statistical analysis. He says that the data show a positive correlation between the number of app downloads and the number of app review impressions. But he's not sure whether they would have downloaded it anyway without seeing the review. What response would be most helpful to Sam?

Self-check of Problem Framing and Data Collection Concepts

You could run an experiment to compare the behavior of users who didn't see the review with similar users who did.

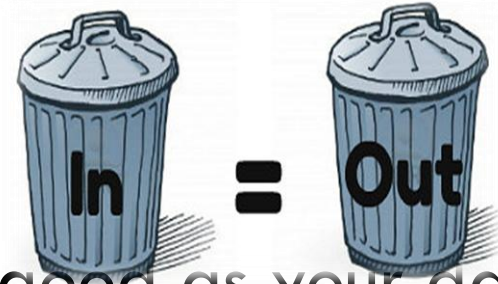
- Correct! If Sam observes that users who saw the positive review were more likely to download the app than those who didn't, then he has reasonable evidence to suggest that the positive review is encouraging people to get the app.

Trust the data. It's clear that that great review is the reason users are downloading the app.

- Incorrect. This response wouldn't lead Sam in the right direction. You can't determine causation from only observational data. Sam is seeing a correlation (that is, a statistical dependency between the numbers) that may or may not indicate causation. Don't let your analyses join the ranks of spurious correlations.

The Size and Quality of a Data Set

- “Garbage in, garbage out”



- After all, your model is only as good as your data.
- But how do you measure your data set's quality and improve it? And how much data do you need to get useful results? The answers depend on the type of problem you're solving.

The Size of a Data Set

- As a rough rule of thumb, your model should train on at least an order of magnitude more examples than trainable parameters. Simple models on large data sets generally beat fancy models on small data sets.

Data set	Size (number of examples)
Iris flower data set	150 (total set)
MovieLens (the 20M data set)	20,000,263 (total set)
Google Gmail SmartReply	238,000,000 (training set)
Google Books Ngram	468,000,000,000 (total set)
Google Translate	trillions

The Quality of a Data Set

- It's no use having a lot of data if it's bad data; quality matters, too.
 - But what counts as "quality"?
 - It's a fuzzy term.
- Consider taking an empirical approach and picking the option that produces the best outcome. With that mindset, a quality data set is one that lets you succeed with the business problem you care about. In other words, the data is *good* if it accomplishes its intended task.

The Quality of a Data Set

- However, while collecting data, it's helpful to have a more concrete definition of quality. Certain aspects of quality tend to correspond to better-performing models:
 - ▣ reliability
 - ▣ feature representation
 - ▣ minimizing skew

Reliability

- **Reliability** refers to the degree to which you can *trust* your data. A model trained on a reliable data set is more likely to yield useful predictions than a model trained on unreliable data. In measuring reliability, you must determine:
 1. How common are label errors? For example, if your data is labeled by humans, sometimes humans make mistakes.
 2. Are your features noisy? For example, GPS measurements fluctuate. Some noise is okay. You'll never purge your data set of *all* noise. You can collect more examples too.
 3. Is the data properly filtered for your problem? For example, should your data set include search queries from bots? If you're building a spam-detection system, then likely the answer is yes, but if you're trying to improve search results for humans, then no.

Reliability

- What makes data unreliable?
- many examples in data sets are unreliable due to one or more of the following:
 - ▣ Omitted values. For instance, a person forgot to enter a value for a house's age.
 - ▣ Duplicate examples. For example, a server mistakenly uploaded the same logs twice.
 - ▣ Bad labels. For instance, a person mislabeled a picture of an oak tree as a maple.
 - ▣ Bad feature values. For example, someone typed an extra digit, or a thermometer was left out in the sun.

Feature Representation

- Representation is the mapping of data to **useful features**. You'll want to consider the following questions:
 - How is data shown to the model?
 - Should you normalize numeric values?
 - How should you handle outliers?

Training versus Prediction

- ❑ Let's say you get great results offline. Then in your live experiment, those results don't hold up. What could be happening?
- ❑ This problem suggests **training/serving skew**—that is, different results are computed for your metrics at training time vs. serving time.
- ❑ Causes of skew can be subtle but have deadly effects on your results. Always consider what data is available to your model at prediction time. During training, use only the features that you'll have available in serving, and make sure your training set is representative of your serving traffic.



Identifying Labels and Sources

Direct vs. Derived Labels

- Machine learning is easier when your labels are well-defined. The best label is a **direct label** of what you want to predict. For example, if you want to predict whether a user is a Taylor Swift fan, a direct label would be "User is a Taylor Swift fan."
- A simpler test of fanhood might be whether the user has watched a Taylor Swift video on YouTube. The label "user has watched a Taylor Swift video on YouTube" is a **derived label** because it does not directly measure what you want to predict. Is this derived label a reliable indicator that the user likes Taylor Swift? Your model will only be as good as the connection between your derived label and your desired prediction.

Label Sources

- The output of your model could be either an Event or an Attribute. This results in the following two types of labels:
- **Direct label for Events**, such as “Did the user click the top search result?”
- **Direct label for Attributes**, such as “Will the advertiser spend more than \$X in the next week?”

Why Use Human Labeled Data?

- There are advantages and disadvantages to using human-labeled data.
- +
 - ▣ Human raters can perform a wide range of tasks.
 - ▣ The data forces you to have a clear problem definition.
- -
 - ▣ The data is expensive for certain domains.
 - ▣ Good data typically requires multiple iterations.

Sampling and Splitting Data

- It's often a struggle to gather enough data for a machine learning project. Sometimes, however, there is *too much* data, and you must select a subset of examples for training.

Sampling and Splitting Data

- How do you select that subset?
- As an example, consider Google Search. At what granularity would you sample its massive amounts of data? Would you use random queries? Random sessions? Random users?

Sampling and Splitting Data

- Ultimately, the answer depends on the problem: what do we want to predict, and what features do we want?
- To use the feature *previous query*, you need to sample at the session level, because sessions contain a sequence of queries.
- To use the feature *user behavior from previous days*, you need to sample at the user level.

Imbalanced Data

- A classification data set with skewed class proportions is called imbalanced. Classes that make up a large proportion of the data set are called majority classes. Those that make up a smaller proportion are minority classes.

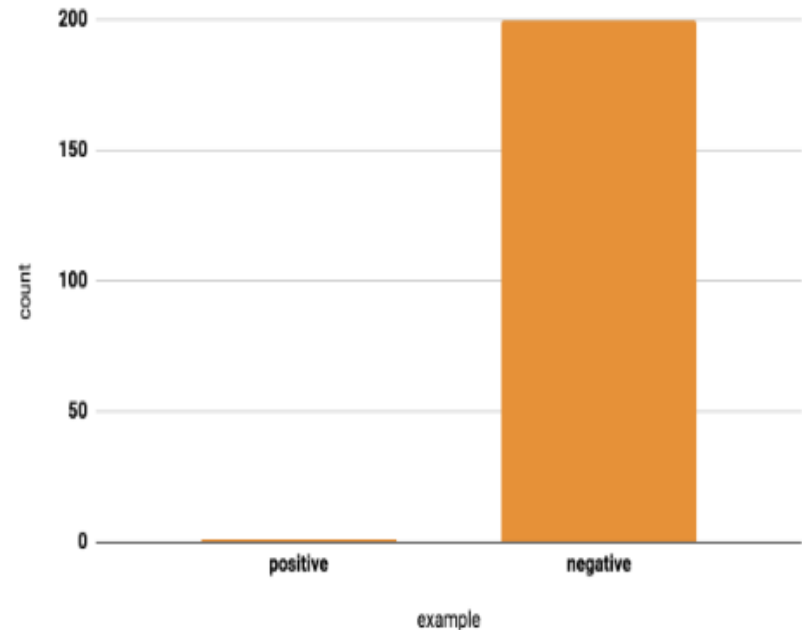
Imbalanced Data

- What counts as imbalanced? The answer could range from mild to extreme, as the table below shows.

Degree of imbalance	Proportion of Minority Class
Mild	20-40% of the data set
Moderate	1-20% of the data set
Extreme	<1% of the data set

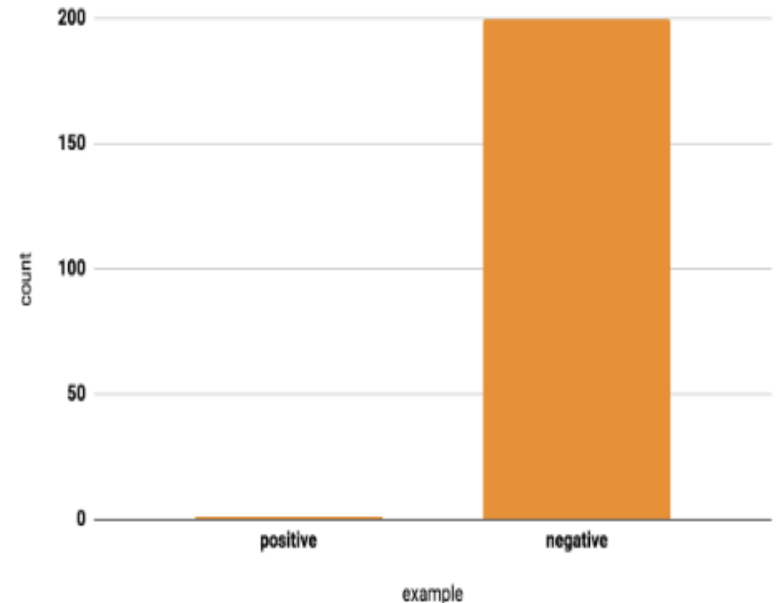
Why look out for imbalanced data?

- Consider the following example of a model that detects fraud. Instances of fraud happen once per 200 transactions in this data set, so in the true distribution, about 0.5% of the data is positive.



Why look out for imbalanced data?

- Why would this be problematic?
- With so few positives relative to negatives, the training model will spend most of its time on negative examples and not learn enough from positive ones.
- For example, if your batch size is 128, many batches will have no positive examples, so the gradients will be less informative.



Why look out for imbalanced data?

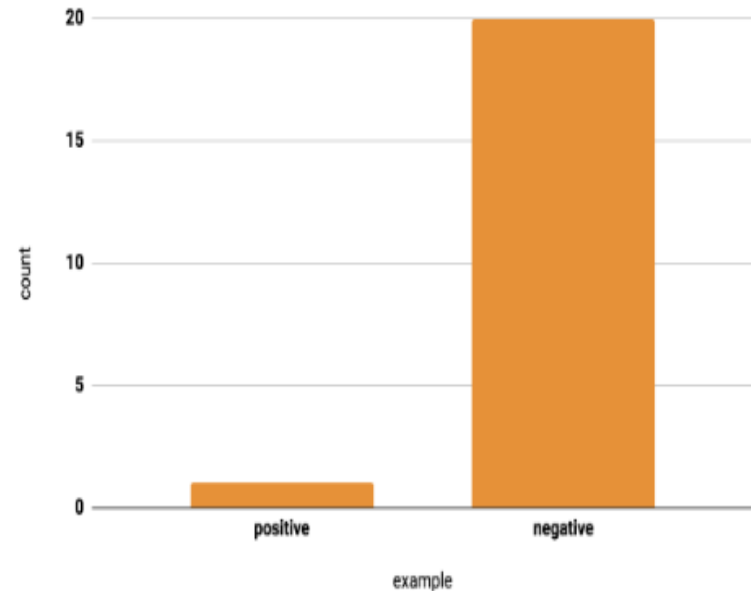
- If you have an imbalanced data set, **first try training on the true distribution**. If the model works well and generalizes, you're done! If not, try the following downsampling and upweighting technique.

Downsampling and Upweighting

- An effective way to handle imbalanced data is to downsample and upweight the majority class. Let's start by defining those two new terms:
 - ▣ **Downsampling** (in this context) means training on a disproportionately low subset of the majority class examples.
 - ▣ **Upweighting** means adding an example weight to the downsampled class equal to the factor by which you downsampled.

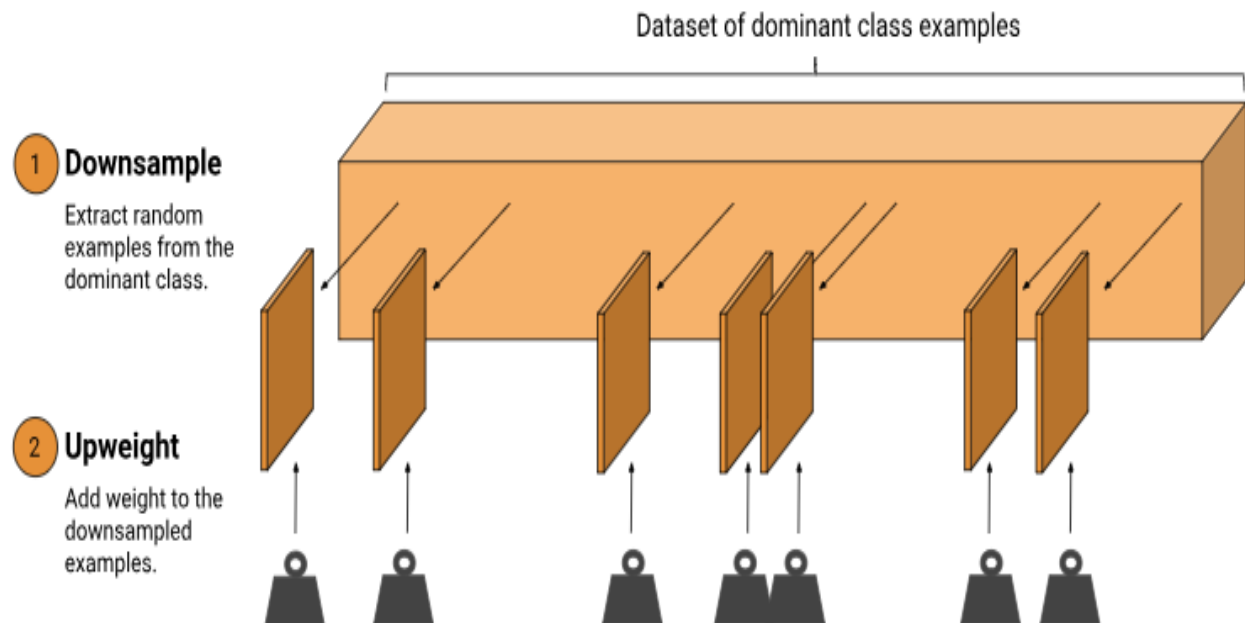
Downsample the majority class

- **Step 1: Downsample the majority class.** Consider again our example of the fraud data set, with 1 positive to 200 negatives. We can downsample by a factor of 20, taking 1/10 negatives. Now about 10% of our data is positive, which will be much better for training our model.



Upweight the downsampled class

- **Step 2: Upweight the downsampled class:** The last step is to add example weights to the downsampled class. Since we downsampled by a factor of 20, the example weight should be 20.



weights

- Here we're talking about *example weights*, which means counting an individual example more importantly during training. An example weight of 10 means the model treats the example as 10 times as important (when computing loss) as it would an example of weight 1.
- The weight should be equal to the factor you used to downsample:

$$\{\text{example weight}\} = \{\text{original example weight}\} \times \{\text{downsampling factor}\}$$

Why Downsample and Upweight?

- It may seem odd to add example weights after downsampling. We were trying to make our model improve on the minority class -- why would we upweight the majority? These are the resulting changes:
 - ▣ **Faster convergence:** During training, we see the minority class more often, which will help the model converge faster.
 - ▣ **Disk space:** By consolidating the majority class into fewer examples with larger weights, we spend less disk space storing them. This savings allows more disk space for the minority class, so we can collect a greater number and a wider range of examples from that class.
 - ▣ **Calibration:** Upweighting ensures our model is still calibrated; the outputs can still be interpreted as probabilities.

Data Split Example

- After collecting your data and sampling where needed, the next step is to split your data into training sets, validation sets, and testing sets.

When Random Splitting isn't the Best Approach

- While random splitting is the best approach for many ML problems, it isn't always the right solution. For example, consider data sets in which the examples are naturally clustered into similar examples.

When Random Splitting isn't the Best Approach

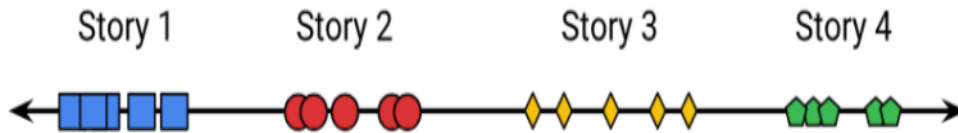


Figure 1. News Stories are Clustered.

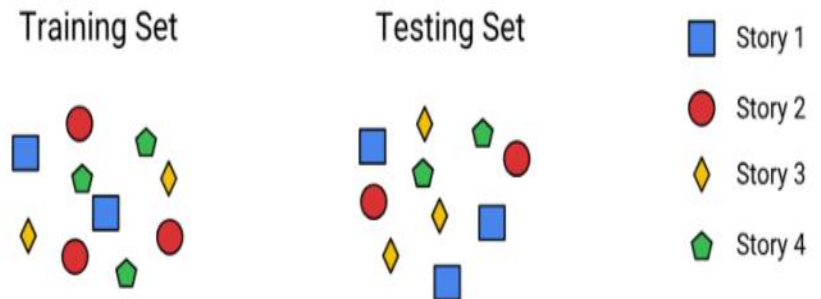


Figure 2. A random split will split a cluster across sets, causing skew.

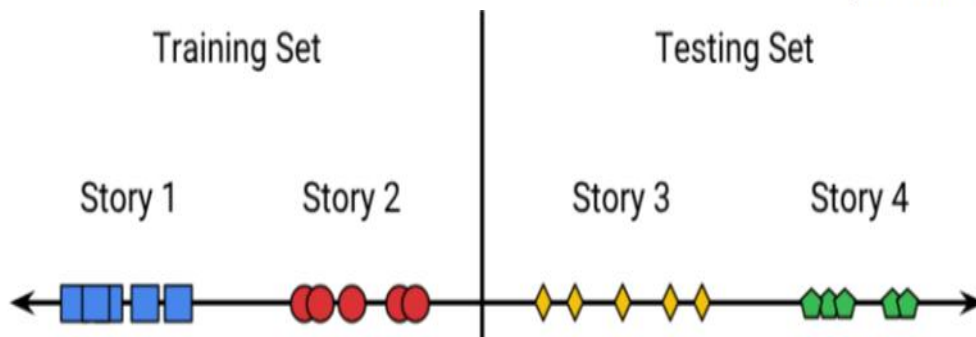
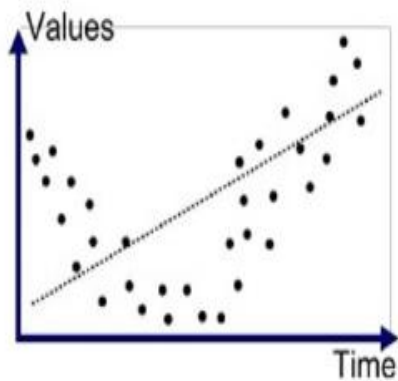
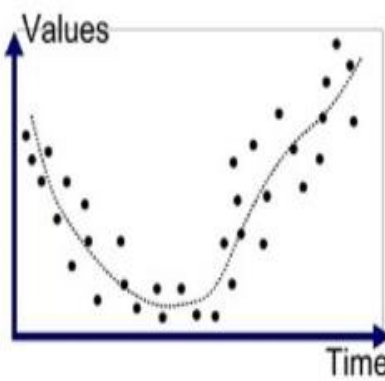


Figure 3. Splitting on time allows the clusters to mostly end up in the same set.

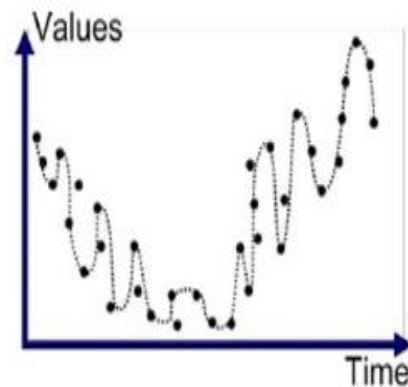
Why Our model perform poor sometime?



Underfitted



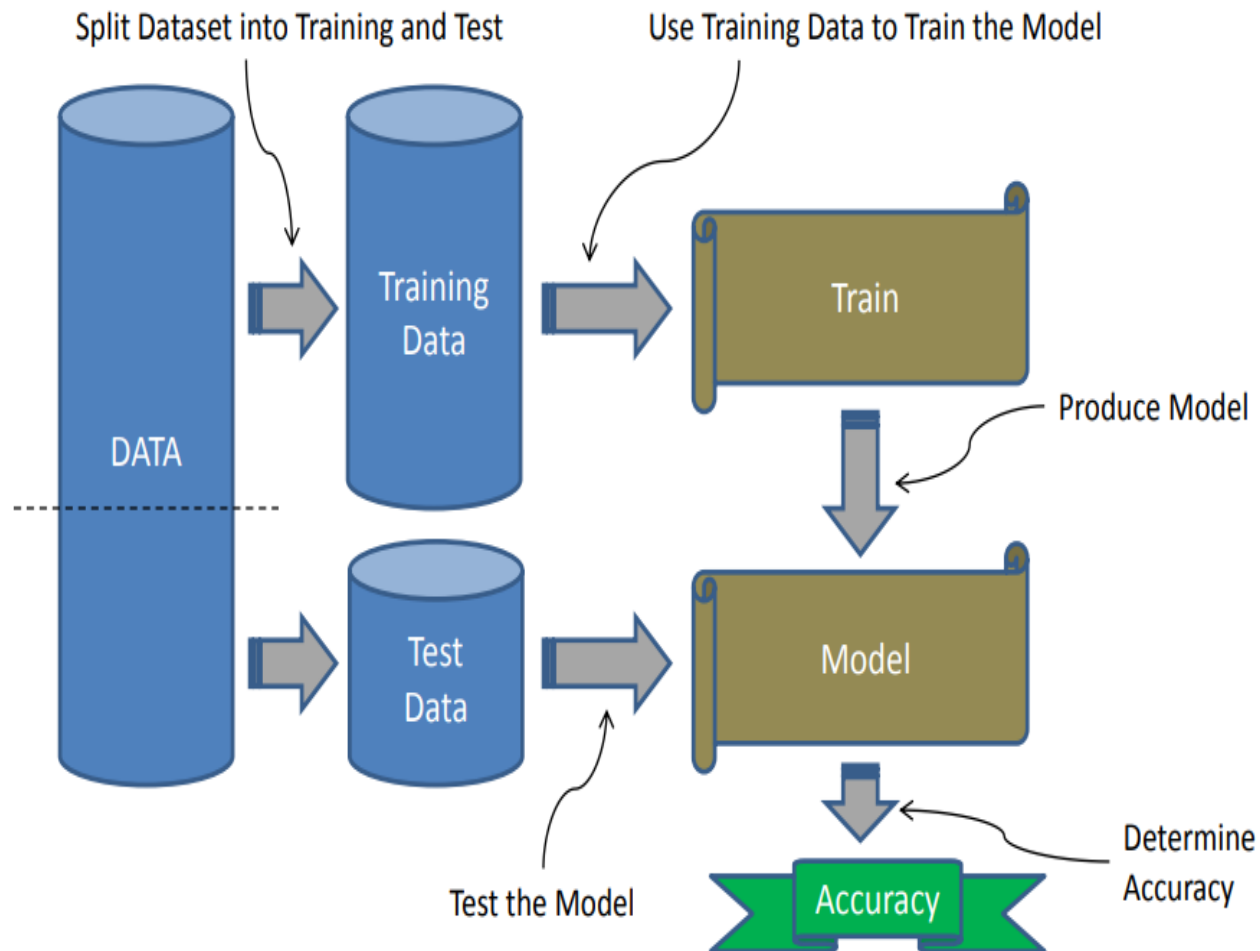
Good Fit/Robust



Overfitted

Image Source: <http://blog.algotrading101.com/design-theories/what-is-curve-fitting-overfitting-in-trading/>

Recall ...



imbalanced data - overfitting

- If the training data is overly unbalanced, then the model will predict a non-meaningful result.
 - For example, if the model is a binary classifier (e.g., cat vs. dog), and nearly all the samples are of the same label (e.g., cat), then the model will simply learn that everything is a that label (cat).
- This is called **overfitting**. To prevent overfitting, there needs to be a fairly equal distribution of training samples for each classification, or range if label is a real value.

Overfitting

- In data science courses, an **overfit** model is explained as having high variance and low bias on the training set which leads to poor generalization on new testing data.

How To Limit Overfitting

- Both overfitting and underfitting can lead to poor model performance. But by far the most common problem in applied machine learning is overfitting.
- Overfitting is such a problem because the evaluation of machine learning algorithms on training data is different from the evaluation we actually care the most about, namely how well the algorithm performs on unseen data.

How To Limit Overfitting

- There are two important techniques that you can use when evaluating machine learning algorithms to limit overfitting:
 1. Use a resampling technique to estimate model accuracy.
 2. Hold back a validation dataset.

Underfitting

- Underfitting refers to a model that can neither model the training data nor generalize to new data.
- An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data.

Basic

- Data science may seem complex but it is really built out of a series of basic building blocks:
 - ▣ **Overfitting:** too much reliance on the training data
 - ▣ **Underfitting:** a failure to learn the relationships in the training data
 - ▣ **High Variance:** model changes significantly based on training data
 - ▣ **High Bias:** assumptions about model lead to ignoring training data
 - ▣ Overfitting and underfitting cause poor **generalization** on the test set
 - ▣ A **validation set** for model tuning can prevent under and overfitting
 - ▣ ...



Web Scrapping

Web Scrapping

API

- **Beautiful Soup**

- ▣ is a tool which help programmer quickly extract valid data from web pages

- **Scrapy**

- ▣ is a web crawling framework

Example



Features engineering

what is **Feature Engineering?**

- Simply put, it is the art/science of **representing data** is the best way possible.
- Good Feature Engineering involves an elegant blend of domain knowledge, intuition, and basic mathematical abilities.

what 'best'?

- In essence, the way you present your data to your algorithm should denote the pertinent **structures/properties** of the underlying *information* in the most effective way possible. When you do feature engineering, you are essentially converting your data *attributes* into data *features*.

Attributes

- Attributes are basically all the dimensions present in your data. But do all of them, in the raw format, represent the underlying trends you want to learn in the best way possible? Maybe not.

Feature

- So what you do in feature engineering, is **pre-process** your data so that your model/learning algorithm has to spend minimum effort on wading through noise.
- ‘noise’ is any information that is not relevant to learning/predicting your ultimate goal.
- In fact, using good features can even let you use considerably simpler models.

Feature

- as with any technique in Machine Learning, always use validation to make sure that the new features you introduce really do improve your predictions, instead of adding unnecessary complexity to your pipeline.

Example

- ❑ Representing timestamps
- ❑ Decomposing Categorical Attributes
- ❑ Binning/Bucketing
- ❑ Feature Crosses
- ❑ Feature Selection
- ❑ Feature Scaling (data normalization)
- ❑ Feature Extraction

Representing timestamps

- Time-stamp attributes are usually denoted by the EPOCH time or split up into multiple dimensions such as (Year, Month, Date, Hours, Minutes, Seconds)
- But in many applications, a lot of that information is unnecessary.

Representing timestamps

- Consider for example a supervised system that tries to predict traffic levels in a city as a function of Location+Time.
- In this case, trying to learn trends that vary by seconds would mostly be misleading. The year wouldn't add much value to the model as well.

Representing timestamps

- Hours, day and month are probably the only dimensions you need. So when representing the time, try to ensure that your model does require all the numbers you are providing it.
- If your data sources come from different geographical sources, do remember to normalize by time-zones if needed.

Decomposing Categorical Attributes

- Some attributes come as categories instead of numbers.
- For example the value of attribute 'color' is one of {Red, Green, Blue}.
 - ▣ Convert each category into a binary attribute that takes one value out of {0, 1}. (one-hot encoding.)

One hot encoding

- One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.

CompanyName		Categoricalvalue	Price
VW	1	20000	
Acura	2	10011	
Honda	3	50000	
Honda	3	10000	

Binning/Bucketing

- Sometimes, it makes more sense to represent a numerical attribute as a categorical one.
- Example : Consider the problem of predicting whether a person owns a certain item of clothing or not.
 - Age might definitely be a factor here. What is actually more pertinent, is the *Age Group*.
 - So what you could do, is have ranges such as 1-10, 11-18, 19-25, 26-40, etc.

Binning

- *Binning* or grouping data (sometimes called *quantization*) is an important tool in preparing numerical data for machine learning,

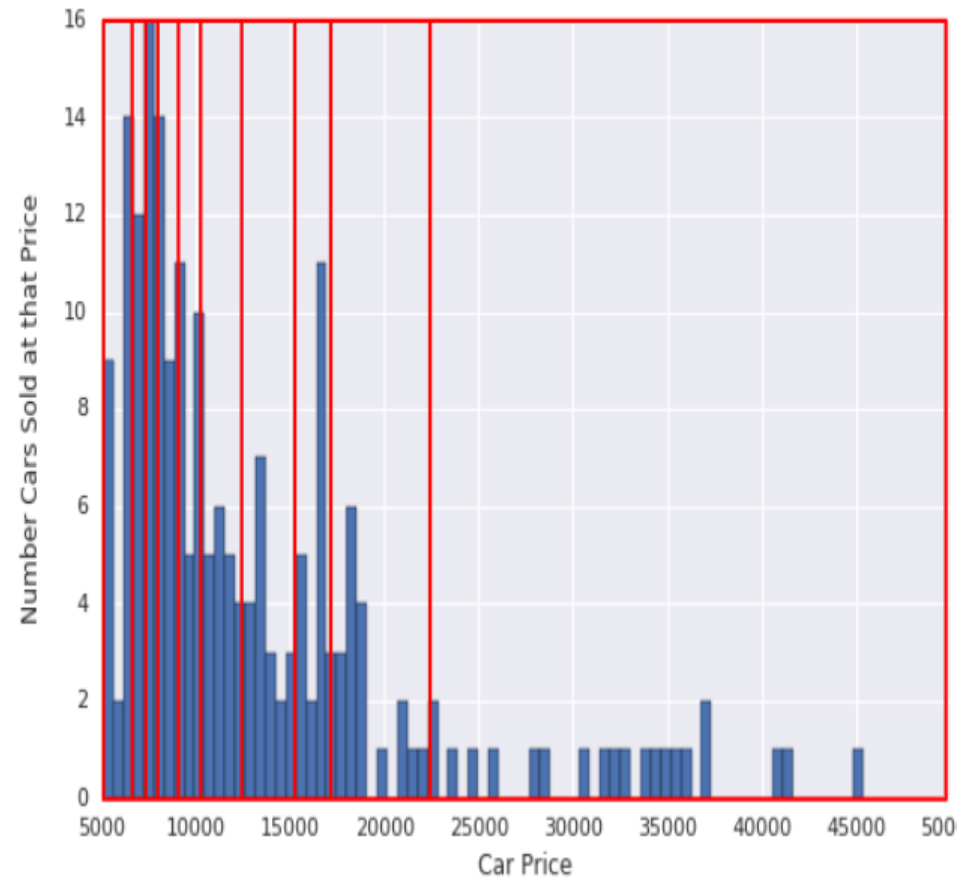
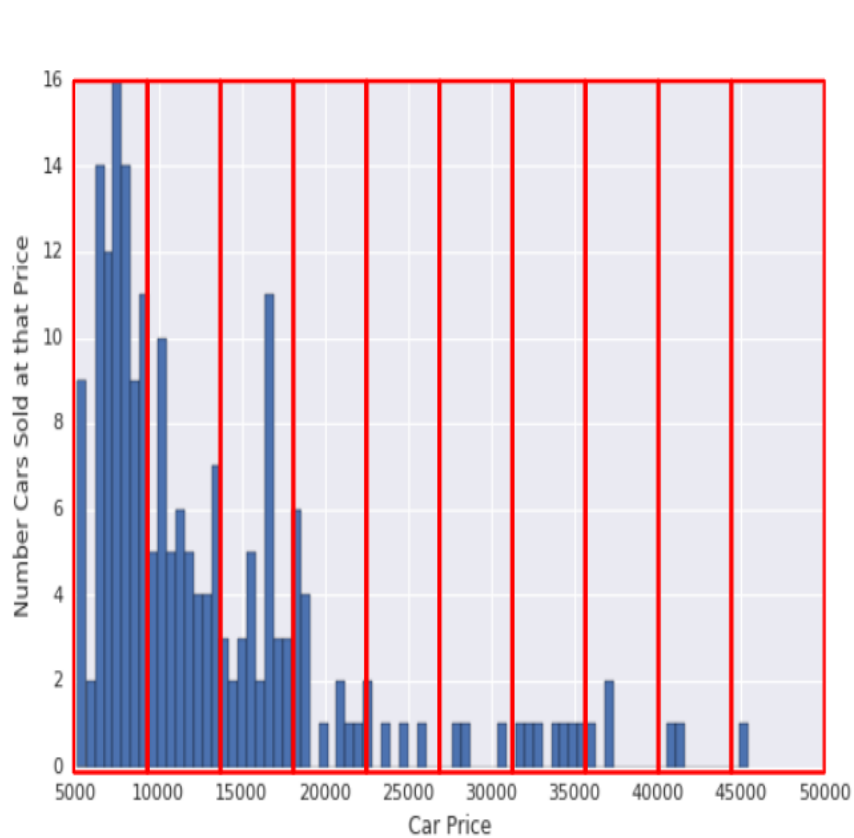
Binning

- Binning also reduces the effect of tiny errors, by 'rounding off' a given value to the nearest representative.
- Binning does *not* make sense if the number of your ranges is comparable to the total possible values, or if precision is very important to you.

Bucketing

- Bucketing makes sense when the domain of your attribute can be divided into **neat ranges**, where all numbers falling in a range imply a **common characteristic**.
 - ▣ It reduces overfitting in certain applications

Example

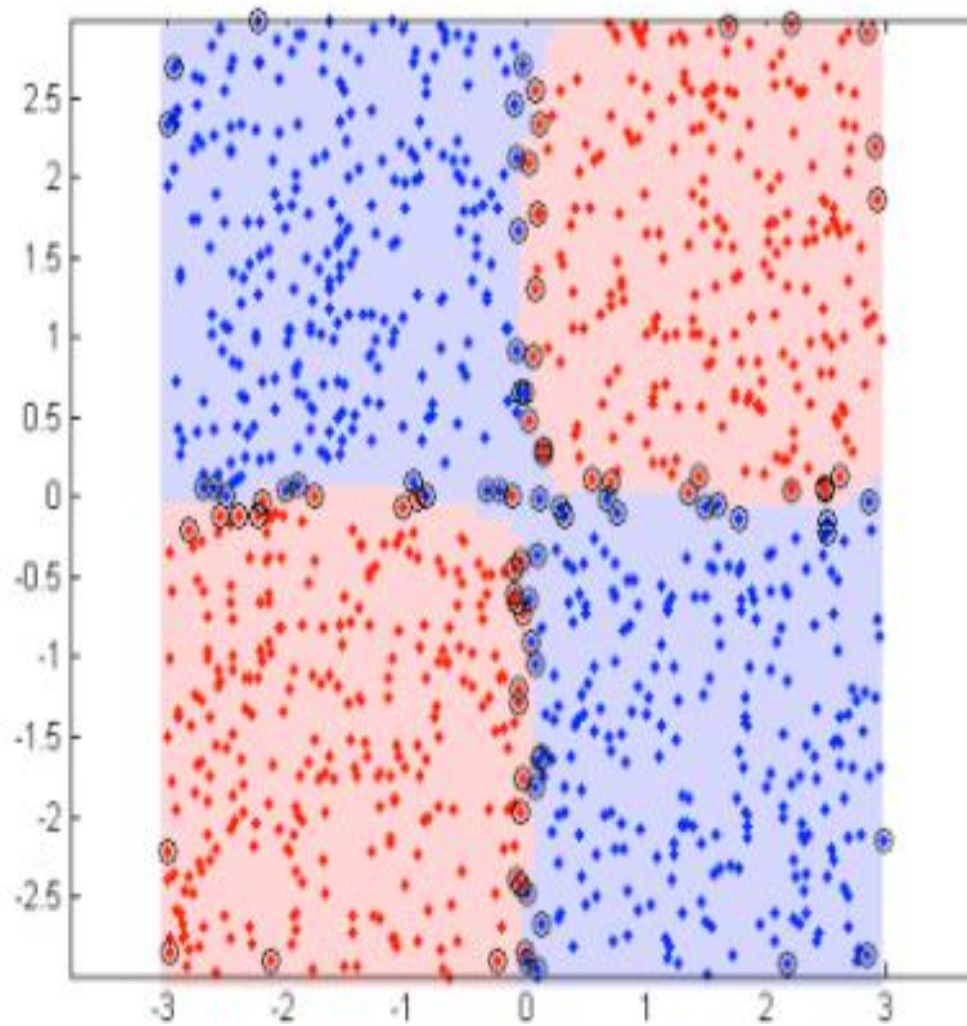


<https://developers.google.com/machine-learning/data-prep/transform/bucketing>

Feature crosses

- **Feature crosses** are a unique way to combine two or more categorical attributes into a *single* one.
- This is extremely useful a technique, when certain features *together* denote a property better than individually by themselves.
 - Mathematically speaking, you are doing a cross product between all possible values of the categorical features.

Example

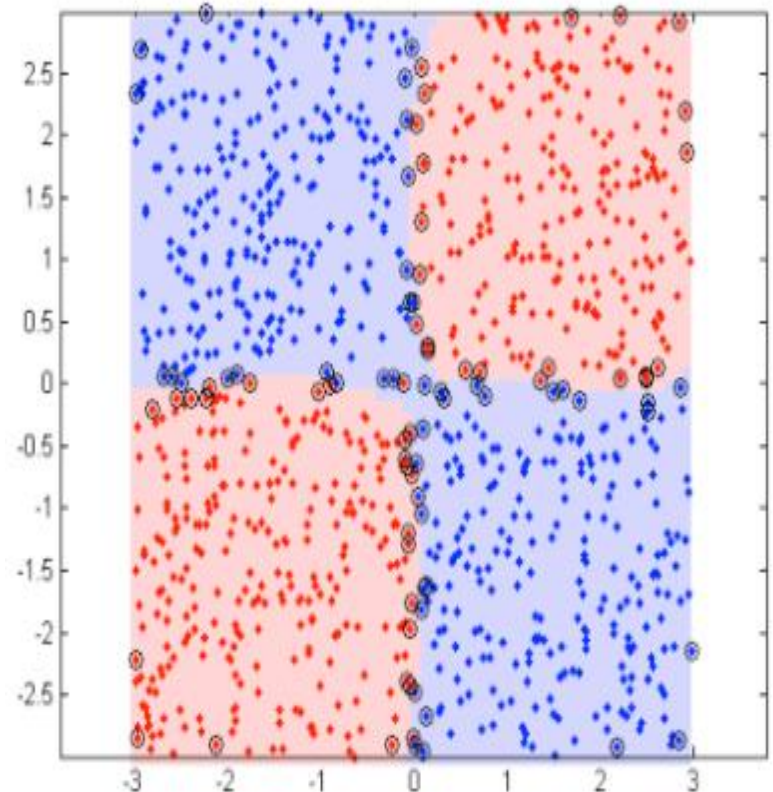


Feature crosses

- Consider a feature A , with two possible values $\{A1, A2\}$. Let B be a feature with possibilities $\{B1, B2\}$. Then, a feature-cross between A & B (lets call it AB) would take one of the following values: $\{(A1, B1), (A1, B2), (A2, B1), (A2, B2)\}$.

Example

- First off, you would benefit from binning the X, Y values into $\{x < 0, x \geq 0\}$ & $\{y < 0, y \geq 0\}$ respectively.
- Lets call them $\{X_n, X_p\}$ and $\{Y_n, Y_p\}$.
- It is pretty obvious that **Quadrants I & III** correspond to class Red, and **Quadrants II & IV** contain class Blue.



So if you could now cross features X and Y into a single feature 'Quadrant', you would basically have $\{I, II, III, IV\}$ being equivalent to $\{(X_p, Y_p), (X_n, Y_p), (X_n, Y_n), (X_p, Y_n)\}$ respectively.

Example

- suppose you define modified features and as follows: $X_{\text{sign}}, Y_{\text{sign}}$
- Now, you could just define a new feature as follows: $X_{\text{sign}} = \frac{x}{|x|}$ and $Y_{\text{sign}} = \frac{y}{|y|}$
-
- $\text{Quadrant}_{\text{odd}} = X_{\text{sign}} Y_{\text{sign}}$
- That's all! If , $\text{Quadrant}_{\text{odd}} = 1$ the class is Red. Else, Blue!

Feature selection

- *Feature selection... is the **process of selecting a subset of relevant** features for use in model construction*
- Feature selection is another key part of the applied machine learning process, like model selection.
- It is important to consider feature selection a part of the model selection process. If you do not, you may inadvertently introduce bias into your models which can result in overfitting.

Feature selection

- Feature selection is different from dimensionality reduction. Both methods seek to reduce the number of attributes in the dataset, but a dimensionality reduction method do so by creating new combinations of attributes, where as feature selection methods include and exclude attributes present in the data without changing them.

Feature Selection

- **Feature Selection** : Using certain algorithms to automatically select a subset of your original features, for your final model.
- Here, you are not creating/modifying your current features, but rather pruning them to reduce noise/redundancy.

Voir: <http://jmlr.csail.mit.edu/papers/volume3/guyon03a/guyon03a.pdf>

Feature selection

- *Feature selection is itself useful, but it mostly acts as a filter, muting out features that aren't useful in addition to your existing features.*
- Feature selection methods can be used to identify and remove unneeded, irrelevant and redundant attributes from data that do not contribute to the accuracy of a predictive model or may in fact decrease the accuracy of the model.

Objective

- *The objective of variable selection is **three-fold**:*
 1. *Improving the prediction performance of the predictors,*
 2. *Providing faster and more cost-effective predictors,*
 3. *and providing a better understanding of the underlying process that generated the data.*

Feature Selection Algorithms

- There are three general classes of feature selection algorithms:
 1. filter methods,
 2. wrapper methods
 3. and embedded methods.

Filter feature selection

- Filter feature selection methods **apply a statistical measure** to assign a **scoring** to each feature.
- The features are ranked by the score and either selected to be kept or removed from the dataset.
- The methods are often univariate and consider the feature independently, or with regard to the dependent variable.
 - ▣ Some examples of some filter methods include the Chi squared test, information gain and correlation coefficient scores.

Wrapper methods

- Wrapper methods consider the selection of a set of features as a **search problem**, where different combinations are prepared, evaluated and compared to other combinations.
- A predictive model is used to **evaluate** a combination of features and assign a score based on model **accuracy**.
- The search process may be methodical such as a best-first search, it may **stochastic** such as a random hill-climbing algorithm, or it may use **heuristics**, like forward and backward passes to add and remove features.
 - ▣ An example of a wrapper method is the recursive feature elimination algorithm.

Embedded Methods

- Embedded methods learn which features best contribute to the **accuracy** of the model while the model is being created.
- The most common type of embedded feature selection methods are **regularization** methods.
- Regularization methods are also called **penalization** methods that introduce additional constraints into the **optimization** of a predictive algorithm (such as a regression algorithm) that bias the model toward lower complexity (fewer coefficients).
 - ▣ Examples of regularization algorithms are the LASSO, Elastic Net and Ridge Regression.

Feature Selection Checklist

1. **Do you have domain knowledge?** If yes, construct a better set of “ad hoc” features
2. **Are your features commensurate?** If no, consider normalizing them.
3. **Do you suspect interdependence of features?** If yes, expand your feature set by constructing conjunctive features or products of features, as much as your computer resources allow you.
4. **Do you need to prune the input variables (e.g. for cost, speed or data understanding reasons)?** If no, construct disjunctive features or weighted sums of feature
5. **Do you need to assess features individually (e.g. to understand their influence on the system or because their number is so large that you need to do a first filtering)?** If yes, use a variable ranking method; else, do it anyway to get baseline results.
6. **Do you need a predictor?** If no, stop

Feature Selection Checklist

7. **Do you suspect your data is “dirty” (has a few meaningless input patterns and/or noisy outputs or wrong class labels)?** If yes, detect the outlier examples using the top ranking variables obtained in step 5 as representation; check and/or discard them.
8. **Do you know what to try first?** If no, use a linear predictor. Use a forward selection method with the “probe” method as a stopping criterion or use the 0-norm embedded method for comparison, following the ranking of step 5, construct a sequence of predictors of same nature using increasing subsets of features. Can you match or improve performance with a smaller subset? If yes, try a non-linear predictor with that subset.
9. **Do you have new ideas, time, computational resources, and enough examples?** If yes, compare several feature selection methods, including your new idea, correlation coefficients, backward selection and embedded methods. Use linear and non-linear predictors. Select the best approach with model selection
10. **Do you want a stable solution (to improve performance and/or understanding)?** If yes, subsample your data and redo your analysis for several “bootstrap”.

Example

- The sklearn.feature_selection module implements feature selection algorithms. It currently includes univariate filter selection methods and the recursive feature elimination algorithm.

sklearn.feature_selection

<code>feature_selection.GenericUnivariateSelect ([...])</code>	Univariate feature selector with configurable strategy.
<code>feature_selection.SelectPercentile ([...])</code>	Select features according to a percentile of the highest scores.
<code>feature_selection.SelectKBest ([score_func, k])</code>	Select features according to the k highest scores.
<code>feature_selection.SelectFpr ([score_func, alpha])</code>	Filter: Select the p-values below alpha based on a FPR test.
<code>feature_selection.SelectFdr ([score_func, alpha])</code>	Filter: Select the p-values for an estimated false discovery rate
<code>feature_selection.SelectFromModel (estimator)</code>	Meta-transformer for selecting features based on importance weights.
<code>feature_selection.SelectFwe ([score_func, alpha])</code>	Filter: Select the p-values corresponding to Family-wise error rate
<code>feature_selection.RFE (estimator[, ...])</code>	Feature ranking with recursive feature elimination.
<code>feature_selection.RFECV (estimator[, step, ...])</code>	Feature ranking with recursive feature elimination and cross-validated selection of the best number of features.
<code>feature_selection.VarianceThreshold ([threshold])</code>	Feature selector that removes all low-variance features.
<code>feature_selection.chi2 (X, y)</code>	Compute chi-squared stats between each non-negative feature and class.
<code>feature_selection.f_classif (X, y)</code>	Compute the ANOVA F-value for the provided sample.
<code>feature_selection.f_regression (X, y[, center])</code>	Univariate linear regression tests.
<code>feature_selection.mutual_info_classif (X, y)</code>	Estimate mutual information for a discrete target variable.
<code>feature_selection.mutual_info_regression (X, y)</code>	Estimate mutual information for a continuous target variable.

Removing features with low variance

- VarianceThreshold is a simple baseline approach to feature selection. It removes all features whose variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e. features that have the same value in all samples.

- Suppose that we have a dataset with boolean features, and we want to remove all features that are either one or zero (on or off) in more than 80% of the samples. Boolean features are Bernoulli random variables, and the variance of such variables is given by: $\text{Var}[X] = p(1 - p)$
- so we can select using the threshold $.8 * (1 - .8)$

```
Entrée [1]:  from sklearn.feature_selection import VarianceThreshold
X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
sel.fit_transform(X)
```

```
Out[1]: array([[0, 1],
               [1, 0],
               [0, 0],
               [1, 1],
               [1, 0],
               [1, 1]])
```


As expected, VarianceThreshold has removed the first column, which has a probability $p=5/6 > .8$ of containing a zero.

The Recursive Feature Elimination (RFE)

- The Recursive Feature Elimination (RFE) method is a feature selection approach. It works by recursively removing attributes and building a model on those attributes that remain. It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.

The Recursive Feature Elimination (RFE)

This recipe shows the use of RFE on the Iris floweres dataset to select 3 attributes.

```
Entrée [5]:  # Recursive Feature Elimination  
from sklearn import datasets  
from sklearn.feature_selection import RFE  
from sklearn.linear_model import LogisticRegression  
# Load the iris datasets  
dataset = datasets.load_iris()  
# create a base classifier used to evaluate a subset of attributes  
model = LogisticRegression()  
# create the RFE model and select 3 attributes  
rfe = RFE(model, 3)  
rfe = rfe.fit(dataset.data, dataset.target)  
# summarize the selection of the attributes  
print(rfe.support_)  
print(rfe.ranking_)
```

```
[False True True True]
```


```
[2 1 1 1]
```

Feature Importance

- Methods that use ensembles of **decision trees** (like Random Forest or Extra Trees) can also compute the **relative importance of each attribute**. These importance values can be used to inform a feature selection process.

Feature Importance

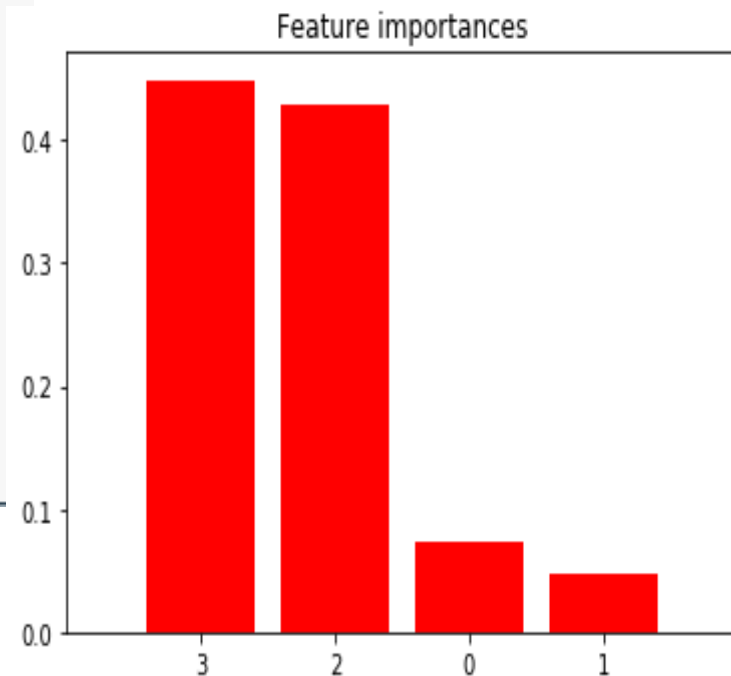
This code shows the construction of an Extra Trees ensemble of the iris flowers dataset and the display of the relative feature importance.

```
Entrée [6]:  # Feature Importance  
from sklearn import datasets  
from sklearn import metrics  
from sklearn.ensemble import ExtraTreesClassifier  
# load the iris datasets  
dataset = datasets.load_iris()  
# fit an Extra Trees model to the data  
model = ExtraTreesClassifier()  
model.fit(dataset.data, dataset.target)  
# display the relative importance of each attribute  
print(model.feature_importances_)
```

```
[0.06237856 0.04333641 0.57610379 0.31818124]
```

Feature Importance

```
importances = model.feature_importances_  
std = np.std([model.feature_importances_ for tree in model.estimators_],  
             axis=0)  
indices = np.argsort(importances)[::-1]  
  
# Print the feature ranking  
print("Feature ranking:")  
X=dataset.data  
for f in range(X.shape[1]):  
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))  
  
# Plot the feature importances of the forest  
plt.figure()  
plt.title("Feature importances")  
plt.bar(range(X.shape[1]), importances[indices],  
        color="r", yerr=std[indices], align="center")  
plt.xticks(range(X.shape[1]), indices)  
plt.xlim([-1, X.shape[1]])  
plt.show()
```



Feature scaling

- **Feature scaling** is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step.

Standardization

- The result of **standardization** (or **Z-score normalization**) is that the features will be rescaled so that they'll have the properties of a standard normal distribution with $\mu=0$ and $\sigma=1$
- where μ is the mean (average) and σ is the standard deviation from the mean; standard scores (also called **z** scores) of the samples are calculated as follows:

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

Standardizing the features

- Standardizing the features so that they are centered around 0 with a standard deviation of 1 is not only important if we are comparing measurements that have different units, but it is also a general requirement for many machine learning algorithms.

Standardizing the features

- with features being on different scales, certain weights may update faster than others since the feature values x_i play a role in the weight updates

- so that
$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_i (t^{(i)} - o^{(i)}) x_j^{(i)},$$

$w_j := w_j + \Delta w_j$, where η is the learning rate, t the target class label, and o the actual output.

Standardizing the features

- Some examples of algorithms where feature scaling matters are:
 - ▣ k-nearest neighbors with an Euclidean distance measure if want all features to contribute equally
 - ▣ k-means (see k-nearest neighbors)
 - ▣ logistic regression, SVMs, perceptrons, neural networks etc. if you are using gradient descent/ascent-based optimization, otherwise some weights will update much faster than others
 - ▣ linear discriminant analysis, principal component analysis, kernel principal component analysis since you want to find directions of maximizing the variance

Min-Max scaling

- In this approach, the data is scaled to a fixed range - usually 0 to 1.
 - ▣ The cost of having this bounded range - in contrast to standardization - is that we will end up with smaller standard deviations, which can suppress the effect of outliers.
- A Min-Max scaling is typically done via the following equation:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Example

- For example, suppose that we have the students' weight data, and the students' weights span [160 pounds, 200 pounds]. To rescale this data, we first subtract 160 from each student's weight and divide the result by 40 (the difference between the maximum and minimum weights).

Mean normalization

$$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

“Standardization or Min-Max scaling?”

- There is no obvious answer to this question: it really **depends on the application.**
 - ▣ For example, in clustering analyses, standardization may be especially crucial in order to compare similarities between features based on certain distance measures
 - ▣ A typical neural network algorithm require data that on a 0-1 scale.



Standardizing and normalizing - how it can be done using scikit-learn

- See Tuto [Scaling.py](#)

DataSet

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Wine Data Set

Download: [Data Folder](#), [Data Set Description](#)

Abstract: Using chemical analysis determine the origin of wines



Data Set Characteristics:	Multivariate	Number of Instances:	178	Area:	Physical
Attribute Characteristics:	Integer, Real	Number of Attributes:	13	Date Donated	1991-07-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	1078813

Source:

Original Owners:

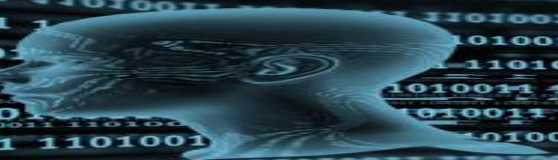
Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno,
16147 Genoa, Italy.

Donor:

Stefan Aeberhard, email: stefan '@' coral.cs.jcu.edu.au

<http://archive.ics.uci.edu/ml/datasets/Wine>

Tuto



Feature Extraction?



Source: <https://deepai.org/machine-learning-glossary-and-terms/feature-extraction>

Feature Extraction: What is ?

- Feature extraction is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing.
- A characteristic of these large data sets is a large number of variables that require a lot of computing resources to process.
 - ▣ Feature extraction is the name for methods that combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set.

What are?

- Dimension Reduction refers to the process of converting a set of data having vast dimensions into data with lesser dimensions ensuring that it conveys similar information concisely.
- These techniques are typically used while solving **machine learning problems** to obtain better features for a classification or regression task.

Dimension reduction techniques

- With more variables, comes more trouble! And to avoid this trouble, **dimension reduction techniques** comes to the rescue.

Human Activity Recognition Using Smartphones Data Set

Download: [Data Folder](#), [Data Set Description](#)

Abstract: Human Activity Recognition database built from the recordings of 30 subjects performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors.

Data Set Characteristics:	Multivariate, Time-Series	Number of Instances:	10299	Area:	Computer
Attribute Characteristics:	N/A	Number of Attributes:	561	Date Donated	2012-12-10
Associated Tasks:	Classification, Clustering	Missing Values?	N/A	Number of Web Hits:	773824

Source:

Jorge L. Reyes-Ortiz(1,2), Davide Anguita(1), Alessandro Ghio(1), Luca Oneto(1) and Xavier Parra(2)

1 - Smartlab - Non-Linear Complex Systems Laboratory

DITEN - Università degli Studi di Genova, Genoa (I-16145), Italy.

2 - CETpD - Technical Research Centre for Dependency Care and Autonomous Living

Universitat Politècnica de Catalunya (BarcelonaTech). Vilanova i la Geltrú (08800), Spain

activityrecognition '@' smartlab.ws

Why is this Useful?

- The process of feature extraction is useful when you need to reduce the number of resources needed for processing without losing important or relevant information.
- Feature extraction can also reduce the amount of redundant data for a given analysis. Also, the reduction of the data and the machine's efforts in building variable combinations (features) facilitate the following learning and generalization steps in the machine learning process.

General dimensionality reduction techniques :

- Independent component analysis
- Isomap
- Kernel PCA
- Latent semantic analysis
- Partial least squares
- Principal component analysis
- Multifactor dimensionality reduction
- Nonlinear dimensionality reduction
- Multilinear Principal Component Analysis
- Multilinear subspace learning
- Semidefinite embedding
- Autoencoder

Missing data

- Missing data in the training data set can reduce the power / fit of a model or can lead to a biased model because we have not analysed the behavior and relationship with other variables correctly. It can lead to wrong prediction or classification.

The common methods to perform Dimension Reduction?

- 1. **Missing Values**: While exploring data, if we encounter missing values, what we do? Our first step should be to identify the reason then impute missing values/ drop variables using appropriate methods.
- But, what if we have too many missing values? Should we impute missing values or drop the variables?

Low Variance

- **Low Variance:** Let's think of a scenario where we have a constant variable (all observations have same value, 5) in our data set. Do you think, it can improve the power of model?
- Ofcourse NOT,because it has zero variance.
 - In case of high number of dimensions, we should drop variables having low variance compared to others because these variables will not explain the variation in target variables.

High Correlation

- **High Correlation:** Dimensions exhibiting higher correlation can lower down the performance of model. Moreover, it is not good to have multiple variables of similar information or variation also known as “**Multicollinearity**”.
- You can use *Pearson* (continuous variables) or *Polychoric* (discrete variables) correlation matrix to identify the variables with high correlation and select one of them using **VIF** (Variance Inflation Factor). Variables having higher value ($VIF > 5$) can be dropped.

Backward Feature Elimination

- **Backward Feature Elimination:** In this method, we start with all n dimensions. Compute the **sum of square of error** (SSR) after eliminating each variable (n times). Then, identifying variables whose removal has produced the smallest increase in the SSR and removing it finally, leaving us with $n-1$ input features.
- Repeat this process until no other variables can be dropped.

Forward Feature Selection

- Reverse to this, we can use “**Forward Feature Selection**” method. In this method, we select one variable and analyse the performance of model by adding another variable. Here, selection of variable is based on higher improvement in model performance.

Factor Analysis

- Let's say some variables are highly correlated. These variables can be grouped by their correlations i.e. all variables in a particular group can be highly correlated among themselves but have low correlation with variables of other group(s).
- Here each group represents a single underlying construct or factor. These factors are small in number as compared to large number of dimensions. However, these factors are difficult to observe. There are basically two methods of performing factor analysis:
 - ▣ EFA (Exploratory Factor Analysis)
 - ▣ CFA (Confirmatory Factor Analysis)

Principal Component Analysis (PCA)

- In this technique, variables are transformed into a new set of variables, which are linear combination of original variables. These new set of variables are known as **principle components**. They are obtained in such a way that first principle component accounts for most of the possible variation of original data after which each succeeding component has the highest possible variance.

Principal Component Analysis (PCA)

- The second principal component must be orthogonal to the first principal component. In other words, it does its best to capture the variance in the data that is not captured by the first principal component. For two-dimensional dataset, there can be only two principal components.

Principal Component Analysis (PCA)

- The principal components are sensitive to the scale of measurement, now to fix this issue we should always standardize variables before applying PCA. Applying PCA to your data set loses its meaning. If interpretability of the results is important for your analysis, PCA is not the right technique for your project.

Tuto

