Machine Learning

Machine learning is a very hot topic for many key reasons, and because it provides the ability to automatically obtain deep insights, recognize unknown patterns, and create high performing predictive models from data, all without requiring explicit programming instructions.

Despite the popularity of the subject, machine learning's true purpose and details are not well understood, except by very technical folks and/or data scientists.

This paper is intended to be a comprehensive, in-depth guide to machine learning, and should be useful to everyone from business executives to machine learning practitioners. It covers virtually all aspects of machine learning (and many related fields) at a high level, and should serve as a sufficient introduction or reference to the terminology, concepts, tools, considerations, and techniques of the field.

This high level understanding is critical if ever involved in a decision-making process surrounding the usage of machine learning, how it can help achieve business and project goals, which machine learning techniques to use, potential pitfalls, and how to interpret the results.

Machine Learning Defined

The oft quoted and widely accepted formal definition of machine learning as stated by field pioneer Tom M. Mitchell is:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E

The following less formal way to describe machine learning.

Machine learning is a subfield of computer science, but is often also referred to as predictive analytics, or predictive modeling. Its goal and usage is to build new and/or leverage existing *algorithms* to *learn* from data, in order to *build generalizable models* that give *accurate predictions*, or to *find patterns*, particularly with *new and unseen similar data*.

Machine Learning Process Overview

Imagine a dataset as a table, where the rows are each *observation* (aka measurement, data point, etc), and the columns for each observation represent the *features* of that observation and their values.

At the outset of a machine learning project, a dataset is usually split into two or three subsets. The minimum subsets are the *training* and *test* datasets, and often an optional third *validation* dataset is created as well.

Once these data subsets are created from the primary dataset, a predictive model or classifier is trained using the training data, and then the model's predictive accuracy is determined using the test data.

As mentioned, machine learning leverages algorithms to automatically model and find patterns in data, usually with the goal of predicting some target output or *response*. These algorithms are heavily based on statistics and mathematical optimization.

Optimization is the process of finding the smallest or largest value (*minima* or *maxima*) of a function, often referred to as a *loss*, or *cost* function in the minimization case. One of the most popular optimization algorithms used in machine learning is called *gradient descent*, and another is known as the *the normal equation*.

In a nutshell, machine learning is all about automatically *learning* a highly accurate predictive or classifier model, or finding unknown patterns in data, by leveraging learning algorithms and optimization techniques.

Types of Learning

The primary categories of machine learning are supervised, unsupervised, and semi-supervised learning. We will focus on the first two in this article.

In supervised learning, the data contains the response variable (*label*) being modeled, and with the goal being that you would like to predict the value or class of the unseen data. Unsupervised learning involves learning from a dataset that has no label or response variable, and is therefore more about finding patterns than prediction.

As i'm a huge NFL and Chicago Bears fan, my team will help exemplify these types of learning! Suppose you have a ton of Chicago Bears data and stats dating from when the team became a chartered member of the NFL (1920) until the present (2016).

Imagine that each row of the data is essentially a team snapshot (or *observation*) of relevant statistics for every game since 1920. The columns in this case, and the data contained in each, represent the *features*(values) of the data, and may include feature data such as game date, game opponent, season wins, season losses, season ending divisional position, post-season berth (Y/N), post-season stats, and perhaps stats specific to the three phases of the game: offense, defense, and special teams.

In the supervised case, your goal may be to use this data to predict if the Bears will win or lose against a certain team during a given game, and at a given field (home or away). Keep in mind that anything can happen in football in terms of pre and game-time injuries, weather conditions, bad referee calls, and so on, so take this simply as an example of an application of supervised learning with a yes or no response (prediction), as opposed to determining the probability or likelihood of 'Da Bears' getting the win.

Since you have historic data of wins and losses (the response) against certain teams at certain football fields, you can leverage supervised learning to create a model to make that prediction.

Now suppose that your goal is to find patterns in the historic data and learn something that you don't already know, or group the team in certain ways throughout history. To do so, you run an unsupervised machine learning algorithm that clusters (groups) the data automatically, and then analyze the clustering results.

With a bit of analysis, one may find that these automatically generated clusters seemingly groups the team into the following example categories over time:

- Strong defense, weak running offense, strong passing offense, weak special teams, playoff berth
- Strong defense, strong running offense, weak passing offense, average special teams, playoff berth
- Weak defense, strong all-around offense, strong special teams, missed the playoffs
- and so on

An example of unsupervised cluster analysis would be to find a potential reason why they missed the playoffs in the third cluster above. Perhaps due to the weak defense? Bears have traditionally been a strong defensive team, and some say that defense wins championships. Just saying…

In either case, each of the above classifications may be found to relate to a certain time frame, which one would expect. Perhaps the team was characterized by one of these groupings more than once throughout their history, and for differing periods of time.

To characterize the team in this way without machine learning techniques, one would have to pour through all historic data and stats, manually find the patterns and assign the classifications (clusters) for every year taking all data into account, and compile the information. That would definitely not be a quick and easy task.

Alternatively, you could write an explicitly coded program to pour through the data, and that has to know what team stats to consider, what thresholds to take into account for each stat, and so forth. It would take a substantial amount of time to write the code, and different programs would need to be written for every problem needing an answer.

Or… you can employ a machine learning algorithm to do all of this automatically for you in a few seconds.

Machine Learning Goals and Outputs

Machine learning algorithms are used primarily for the following types of output:

- Clustering (Unsupervised)
- Two-class and multi-class classification (Supervised)
- Regression: Univariate, Multivariate, etc. (Supervised)
- Anomaly detection (Unsupervised and Supervised)
- Recommendation systems (aka recommendation engine)

Specific algorithms that are used for each output type are discussed in the next section, but first, let's give a general overview of each of the above output, or problem types.

As discussed, clustering is an unsupervised technique for discovering the composition and structure of a given set of data. It is a process of clumping data into clusters to see what groupings emerge, if any. Each cluster is characterized by a contained set of data points, and a cluster *centroid*. The cluster centroid is basically the mean (average) of all of the data points that the cluster contains, across all features.

Classification problems involve placing a data point (aka observation) into a pre-defined class or category. Sometimes classification problems simply assign a class to an observation, and in other cases the goal is to estimate the probabilities that an observation belongs to each of the given classes.

A great example of a two-class classification is assigning the class of *Spam* or *Ham* to an incoming email, where ham just means 'not spam'. Multi-class classification just means more than two possible classes. So in the spam example, perhaps a third class would be 'Unknown'.

Regression is just a fancy word for saying that a model will assign a continuous value (response) to a data observation, as opposed to a discrete class. A great example of this would be predicting the closing price of the Dow Jones Industrial Average on any given day. This value could be any number, and would therefore be a perfect candidate for regression.

Note that sometimes the word regression is used in the name of an algorithm that is actually used for classification problems, or to predict a discrete categorical response (e.g., spam or ham). A good example is *logistic regression*, which predicts probabilities of a given discrete value.

Another problem type is *anomaly detection*. While we'd love to think that data is well behaved and sensible, unfortunately this is often not the case. Sometimes there are erroneous data points due to malfunctions or errors in measurement, or sometimes due to fraud. Other times it could be that anomalous measurements are indicative of a failing piece of hardware or electronics.

Sometimes anomalies are indicative of a real problem and are not easily explained, such as a manufacturing defect, and in this case, detecting anomalies provides a measure of quality control, as well as insight into whether steps taken to reduce defects have worked or not. In either case, there are times where it is beneficial to find these anomalous values, and certain machine learning algorithms can be used to do just that.

The final type of problem is addressed with a *recommendation system*, or also called *recommendation engine*. Recommendation systems are a type of *information filtering system*, and are intended to make recommendations in many applications, including movies, music, books, restaurants, articles, products, and so on. The two most common approaches are *content-based* and *collaborative* filtering.

Two great examples of popular recommendation engines are those offered by Netflix and Amazon. Netflix makes recommendations in order to keep viewers engaged and supplied with plenty of content to watch. In other words, to keep people using Netflix. They do this with their *"Because you watched …"*, *"Top Picks for Alex"*, and *"Suggestions for you"* recommendations.

Amazon does a similar thing in order to increase sales through up-selling, maintain sales through user engagement, and so on. They do this through their *"Customers Who Bought This Item Also Bought"*, *"Recommendations for You, Alex"*, *"Related to Items You Viewed"*, and *"More Items to Consider"*recommendations.

Machine Learning Algorithms

We've now covered the machine learning problem types and desired outputs. Now we will give a high level overview of relevant machine learning algorithms.

Here is a list of algorithms, both supervised and unsupervised, that are very popular and worth knowing about at a high level. Note that some of these algorithms will be discussed in greater depth later in this series.

Supervised Regression

- Simple and multiple linear regression
- Decision tree or forest regression
- Artificial Neural networks
- Ordinal regression
- Poisson regression
- Nearest neighbor methods (e.g., k-NN or k-Nearest Neighbors)

Supervised Two-class & Multi-class Classification

- Logistic regression and multinomial regression

- Artificial Neural networks
- Decision tree, forest, and jungles
- SVM (support vector machine)
- Perceptron methods
- Bayesian classifiers (e.g., Naive Bayes)
- Nearest neighbor methods (e.g., k-NN or k-Nearest Neighbors)
- One versus all multiclass

Unsupervised

- K-means clustering
- Hierarchical clustering

Anomaly Detection

- Support vector machine (one class)
- PCA (Principle component analysis)

Note that a technique that's often used to improve model performance is to combine the results of multiple models. This approach leverages what's known as *ensemble methods*, and *random forests* are a great example (discussed later).

If nothing else, it's a good idea to at least familiarize yourself with the names of these popular algorithms, and have a basic idea as to the type of machine learning problem and output that they may be well suited for.

Model Performance Introduction

Model *performance* can be defined in many ways, but in general, it refers to how effectively the model is able to achieve the solution goals for a given problem (e.g., prediction, classification, anomaly detection, recommendation).

Since the goals can differ for each problem, the measure of performance can differ as well. Some common performance measures include *accuracy*, *precision*, *recall*, *receiver operator characteristic (ROC)*, and so on. These will be discussed in much greater detail throughout the rest of this series.

Data Selection and Preprocessing

Some say that *garbage in equals garbage out*, and this is definitely the case. This basically means that you may have built a predictive model, but it doesn't matter if the data used to build the model is non-representative, low quality, error ridden, and so on. The quality, amount, preparation, and selection of data is critical to the success of a machine learning solution.

The first step to ensure success is to avoid *selection bias*. Selection bias occurs when the samples used to produce the model are not fully representative of cases that the model may be used for in the future, particularly with new and unseen data.

Data is typically messy and often consists of missing values, useless values (e.g., NA), outliers, and so on. Prior to modeling and analysis, raw data needs to be parsed, cleaned, transformed, and pre-processed. This is typically referred to a *data munging* or *data wrangling*.

For missing data, data is often *imputed*, which is a technique used to fill in, or substitute for missing values, and is very similar conceptually to *interpolation*.

In addition, sometimes feature values are scaled (*feature scaling*) and/or *standardized* (*normalized*). The most typical method of standardizing feature data is to subtract the mean across a given feature's values from each individual observation value, and then divide by the standard deviation of that feature's values.

Feature scaling is used to bring the different feature's value ranges into similarity in order to help prevent certain features from dominating models and predictions, but also to prevent computing problems when running machine learning optimization algorithms (speed, convergence, etc.).

Another preprocessing technique is to create *dummy variables*, which basically means that you convert qualitative variables to quantitative variables. An example is taking a color feature (e.g., green, red, and blue), and transforming it to the values 1, 2, and 3 respectively. This makes it possible to perform regression with qualitative features.

Data Splitting

Recall from the first article that the data used for machine learning should be split into training and test datasets, as well as an optional third validation dataset for model validation and tuning.

Choosing the size of each data set can be somewhat subjective and dependent on the overall sample size, and a full discussion is out of scope for this series. As an example however, given a training and test dataset only, some people may split the data into 80% training and 20% testing.

In general, more *training data* results in a better model and potential performance, and more *testing data*results in a greater evaluation of model performance and overall generalization capability.

Feature Selection and Feature Engineering

Once you have a representative, unbiased, cleaned, and fully prepared dataset, typical next steps include *feature selection* and *feature engineering* of the training data. Note that although discussed here, both of these techniques can also be used later in the process for improving model performance.

Feature selection is the process of selecting a subset of features from which to build a predictive regression model or classifier. This is usually done for model simplification and increased *interpretability*, reducing training times and *computational cost*, and to help reduce the risk of overfitting, and thus improve model generalization.

Basic techniques for feature selection, particularly for regression problems, involve estimates of model *parameters* (i.e., model *coefficients*) and their significance, and *correlation* estimates amongst features. This will be discussed further in a section about *parametric models*.

Some advanced techniques used for feature selection are *principle component analysis* (*PCA*), *singular value decomposition* (*SVD*), and *Linear Discriminant Analysis* (*LDA*).

Principal component analysis is a statistical technique that deals with determining which features, in order, represent the most to least variance in the data. Singular value decomposition is a lower level linear algebra algorithm that is used by PCA.

Linear discriminant analysis is closely related to PCA in that they're both linear transformation techniques. PCA however is more general and is not concerned with class labels (unsupervised), whereas LDA is more specific and is concerned with class labels (supervised).

Feature engineering includes feature selection as a sub-category, but also involves other aspects such as creating new features, transforming raw data into domain-specific and interpretable features, and so on.

Parametric Models and Feature Selection

Many machine learning models are a type of *parametric model*. A good example is the equation describing a line (i.e., linear model), which is shown here, and includes the slope (β), intercept coefficient (α), and an error term (ε).

$$y_i = \alpha + \beta x_i + \varepsilon_i.$$

With parametric models, the coefficients of the terms are called the *parameters*, and are usually designated by the Greek letter *beta* and a subscript (e.g., $\beta_1 \ldots \beta_n$). In regression problems, the parameters are called *regression coefficients*.

Many models also include an error term, indicated by the Greek letter *epsilon*. Simply stated, this error term is meant to account for the difference between the model's *predicted value* and the actual *observed value* for a given set of input values.

Understanding the concept of model parameters is very important for supervised learning because machine learning differs from other techniques, in that it *learns* model parameters automatically. It does this by *estimating* the optimal set of model parameters that best explains the relationship between the response variable and the independent feature variables through *optimization* techniques, as discussed in the first article.

In regression problems, a *p-value* is assigned to each of the estimated model parameters (*regression coefficients*), and this value is used to indicate the potential predictive influence that each coefficient has on the response.

Coefficients with a p-value greater than some chosen threshold, typically 0.05 or 0.10, are often not included in the model since they will most likely not help explain (predict) the response. This is one key way to perform feature selection with parametric models.

Another technique involves estimating the *correlation* of the features with respect to the response, and removing redundant and highly correlated features. The idea is that including only one of a pair of correlated features (the most significant) should be enough to explain the impact of both of the correlated features on the response.

Model Selection

While the algorithm or model that you choose may not matter as much as other things discussed in this series (e.g., amount of data, feature selection, etc.), here is a list of things to take into account when choosing a model.

- Interpretability
- Simplicity (aka *parsimony*)
- Accuracy
- Speed (training, testing, and real-time processing)

- Scalability

A good approach is to start with simple models and then increase model complexity as needed, and only when necessary. Generally, simplicity should be preferred unless you can achieve major accuracy gains through model selection.

Relatively simple models include simple and multiple linear regression for regression problems, and logistic and multinomial regression for classification problems.

A basic early model selection choice for supervised learning is whether to use a linear or nonlinear model. Nonlinear models best describe and predict situations when the effects on the response from certain feature values and their combination is nonlinear. Most of the time however, relationships are never truly linear.

Beyond basic linear models, variations in the response variable can also be due to interaction effects, which means that the response is dependent not only on certain individual features (*main effects*), but also on the combination of certain features (*interaction effects*). This combination of features in a model is represented by multiplying the feature values for each *interaction term* in the model (e.g., $\beta x_1 x_2$) with a term coefficient.

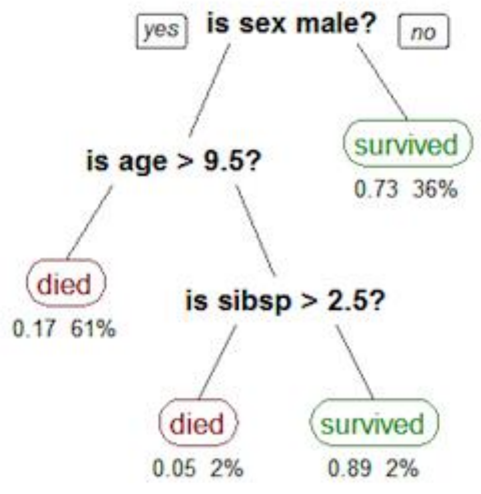
Once interaction terms are included, the significance of the interactions in explaining the response, and whether to include them, can be determined through the usual methods such as p-value estimation. Note that there is a concept known as the *hierarchy principle*, which basically says that if an interaction is included in a model, the associated main effects should also be included.

While linear assumptions are often good enough and can produce adequate results, most real life feature/response relationships are nonlinear, and sometimes nonlinear models are required to get an acceptable level of accuracy. In this case, there are a wide variety of models to choose from.

Nonlinear models can include different degree *polynomials*, *step functions*, *piecewise polynomials*, *splines*, *local regression* (aka *LOESS* models), and *generalized additive models* (GAM). Due to the technical nature of nonlinear modeling, familiarity with the above model approaches by name should suffice for the purpose of this series.

Other notable model choices include *decision trees*, *support vector machines* (*SVM*), and *artificial neural networks* (modeled after biological neural networks, an interconnected system of *neurons*). Decision trees can be highly interpretable, while the latter two are black box and very complex technical methods. Decision trees involve creating a series of splits based on logical decisions, starting from the most important top-level node. Decision trees visually look like an upside down tree.

Here is an example of a decision tree created by Stephen Milborrow, which shows survival of passengers on board the Titanic. The term ‘sibsp’ is the number of spouses or siblings aboard, and the numbers under each leaf refer to the probability of survival and the percentage of the total observations (i.e., people on board). So the upper right leaf indicates that females that survived had a 73% chance of survival and represented 36% of those on board.



The final model selection decision discussed here is whether to leverage *ensemble* methods for additional performance gains. These methods combine models to produce a single consensus prediction or classification, and do so through averaging or voting techniques.

Some very common ensemble methods are *bagging*, *boosting*, and *random forests*. Random forests are essentially bagging applied to decision trees, with the additional element of random feature subset selection.

Model Tradeoffs

Model accuracy is determined in many ways, and will be discussed in detail later in this series. The primary measure of model accuracy comes from estimating the test error for a given model. The accuracy improvement goal of model selection is therefore to reduce the estimated test error.

It is important to note that the goal isn't to find the absolute minimal error, but rather to *find the simplest model that performs well enough*. There are usually diminishing returns in trying the squeeze out the very last bit of performance. Given this, your choice of modeling approach won't always be based on the one that results in the greatest degree of accuracy. Sometimes there are other important factors that must be taken into account as well, including *interpretability*, *simplicity*, *speed*, and *scalability*.

Often, it's a tradeoff choosing whether prediction accuracy or model interpretability is more important for a given application. Artificial neural networks, support vector machines, and some ensemble methods can be used to create very accurate predictive models, but are very much of a black box except to highly specialized and technical individuals.

Black box algorithms may be preferred when predictive performance is the most important goal, and it's not necessary to explain how the model works and makes predictions. In some cases however, model interpretability is preferred, and sometimes legally mandatory.

Here is an interpretability-driven example often seen in the financial industry. Suppose a machine learning algorithm is used to accept or reject an individual's credit card application. If the applicant is rejected and decides to file a complaint or take legal action, the financial institution will need to explain how that decision was made. While that can be nearly impossible for a neural network or SVM system, it's relatively straightforward for decision tree-based algorithms.

In terms of training, testing, processing, and prediction speed, some algorithms and model types take more time, and require greater computing power and memory than others. In some applications, speed and scalability are critical factors, particularly in any widely used, near real-time application (e.g., eCommerce site) where a model needs to be updated fairly regularly, and that performs predictions and/or classifications at scale on the fly.

Lastly, and as previously mentioned, model simplicity (or *parsimony*) should always be preferred unless there is a significant and justifiable gain in performance accuracy. Simplicity usually results in quicker, more scalable, and easier to interpret models and results.

Overfitting

Overfitting is one of the greatest concerns in predictive analytics and machine learning. Overfitting refers to a situation where the model chosen to fit the training data fits too well, and essentially captures all of the noise, outliers, and so on.

The consequence of this is that the model will fit the training data very well, but will not accurately predict cases not represented by the training data, and therefore will not generalize well to unseen data. This means that the model performance will be better with the training data than with the test data.

A model is said to have high *variance* when it leans more towards overfitting, and conversely has high *bias*when it doesn't fit the data well enough. A high variance model will tend to be quite flexible and overly complex, while a high bias model will tend to be very opinionated and overly simplified. A good example of a high bias model is fitting a straight line to very nonlinear data.

In both cases, the model will not make very accurate predictions on new data. The ideal situation is to find a model that is not overly biased, nor does it have a high variance. Finding this balance is one of the key skills of a data scientist.

Overfitting can occur for many reasons. A common one is that the training data consists of many features relative to the number of observations or data points. In this case, the data is relatively wide as compared to long.

To address this problem, reducing the number of features can help, or finding more data if possible. The downside to reducing features is that you lose potentially valuable information.

Another option is to use a technique called *regularization*, which will be discussed later in this series.

Controlling Model Complexity

Model complexity can be characterized by many things, and is a bit subjective. In machine learning, model complexity often refers to the number of features or terms included in a given predictive model, as well as whether the chosen model is linear, nonlinear, and so on. It can also refer to the algorithmic learning complexity or computational complexity.

Overly complex models are less easily interpreted, at greater risk of overfitting, and will likely be more computationally expensive.

There are some really sophisticated and automated methods by which to control, and ultimately reduce model complexity, as well as help prevent overfitting. Some of them are able to help with feature and model selection as well.

These methods include linear model and *subset selection*, *shrinkage* methods (including *regularization*), and *dimensionality reduction*.

Regularization essentially keeps all features, but reduces (or penalizes) the effect of some features on the model's predicted values. The reduced effect comes from shrinking the magnitude, and therefore the effect, of some of the model's term's coefficients.

The two most popular regularization methods are *ridge regression* and *lasso*. Both methods involve adding a tuning parameter (Greek *lambda*) to the model, which is designed to impose a penalty on each term's coefficient based on its size, or effect on the model.

The larger the term's coefficient size, the larger the penalty, which basically means the more the tuning parameter forces the coefficient to be closer to zero. Choosing the value to use for the tuning parameter is critical and can be done using a technique such as *cross-validation*.

The lasso technique works in a very similar way to ridge regression, but can also be used for feature selection as well. This is due to the fact that the penalty term for each predictor is calculated slightly differently, and can result in certain terms becoming zero since their coefficients can become zero. This essentially removes those terms from the model, and is therefore a form of automatic feature selection.

Ridge regression or lasso techniques may work better for a given situation. Often the lasso works better for data where the response is best modeled as a function of a small number of the predictors, but this isn't guaranteed. Cross-validation is a great technique for evaluating one technique versus the other.

Given a certain number of predictors (features), there is a calculable number of possible models that can be created with only a subset of the total predictors. An example is when you have 10 predictors, but want to find all possible models using only 2 of the 10 predictors.

Doing this, and then selecting one of the models based on the smallest test error, is known as *subset selection*, or sometimes as *best subset selection*. Note that a very useful plot for subset selection is when plotting the *residual sum of squares* (discussed later) for each model against the number of predictors.

When the number of predictors gets large enough, best subset selection becomes unable to deal with the huge number of possible model combinations for a given subset of predictors. In this case, another method known as *stepwise selection* can be used. There are two primary versions, forward and backward stepwise selection.

In forward stepwise selection, predictors are added to the model one at a time starting at zero predictors, until all of the predictors are included. Backwards stepwise selection is the opposite, and involves starting with a model including all predictors, and then removing a single predictor at each step.

The model performance is evaluated at each step in both cases. In both subset selection and stepwise selection, the test error is used to determine the best model. There are many ways to estimate test errors, which will be discussed later in this series.

There is a concept that deals with highly dimensional data (i.e., large number of features) known as the *curse of dimensionality*. The *curse of dimensionality* refers to the fact that the computational speed and memory required increases exponentially as the number of data dimensions (features) increases.

This can manifest itself as a problem where a machine learning algorithm does not scale well to higher dimensional data. One way to deal with this issue is to choose a different algorithm that can scale better with the data. The other is a technique known as *dimensionality reduction*.

Dimensionality Reduction

Dimensionality reduction is a technique used to reduce the number of features included in the machine learning process. It can help reduce complexity, reduce computational cost, and increase machine learning algorithm computational speed. It can be thought of as a technique that transforms the original predictors to a new, smaller set of predictors, which are then used to fit a model.

Principal component analysis (PCA) was discussed previously in the context of feature selection, but is also a widely-used dimensionality reduction technique as well. It helps reduce the number of features (i.e., dimensions) by finding, separating out, and sorting the features that explain the most variance in the data in descending order. Cross-validation is a great way to determine the number of principal components to include in the model.

An example of this would be a dataset where each observation is described by ten features, but only three of the features can describe the majority of the data's variance, and therefore are adequate enough for creating a model with, and generating accurate predictions.

Note that people sometimes use PCA to prevent overfitting since fewer features implies that the model is less likely to overfit. While PCA may work in this context, it is not a good approach and is therefore not recommended. Regularization should be used to address overfitting concerns instead.

Model Evaluation and Performance

Assuming you are working with high quality, unbiased, and representative data, then the next most important aspects of predictive analytics and machine learning is measuring model performance, possibly improving it if needed, and understanding potential errors that are often encountered.

We will have an introductory discussion here about model performance, improvement, and errors, but will continue with much greater detail on these topics in the next article.

Model performance is typically used to describe how well a model is able to make predictions on unseen data (e.g., test, but NOT training data), and there are multiple methods and metrics used to assess and gauge model performance. A key measure of model performance is to estimate the model's test error.

The test error can be estimated either indirectly or directly. It can estimated and adjusted indirectly by making changes that affect the training error, since the training error is a measure of overfitting (bias and/or variance) to some extent.

Recall that the more the model overfits the data (high variance), the less well the model will generalize to unseen data. Given that, the assumption is that reducing variance should improve the test error as well.

The test error can also be estimated directly by testing the model with the held out test data, and usually works best in conjunction with a *resampling method* such as *cross-validation*, which we'll discuss later.

Estimating a model's test error not only helps determine a model's performance and accuracy, but is also a very powerful way to select a model too.

Improving Model Performance and Ensemble Learning

There are many ways to improve a model’s performance. The quality and quantity of data used has a huge, if not the biggest impact on model performance, but sometimes these two can’t easily be changed.

Other major influencers on model performance include algorithm tuning, feature engineering, cross-validation, and ensemble methods.

Algorithm tuning refers to the process of *tweaking* certain values that effectively initialize and control how a machine learning algorithm *learns* and generates predictive models. This *tuning* can be used to improve performance using the separate validation data set, and later performance tested with the test dataset.

Since most algorithm tuning parameters are algorithm-specific and sometimes very complex, a detailed discussion is out of scope for this article, but note that the lambda parameter described for regularization is one such tuning parameter.

Ensemble learning, as mentioned in an earlier article, deals with combining or averaging (regression) the results from multiple learning models in order to improve predictive performance. In some cases (classification), ensemble methods can be thought of as a voting process where the majority vote wins.

Two of the most common ensemble methods are *bagging* (aka bootstrap aggregating) and *boosting*. Both are helpful with improving model performance and in reducing variance (overfitting) and bias (underfitting).

Bagging is a technique by which the training data is sampled *with replacement* multiple times. Each time a new training data set is created and a model is fitted to the sample data. The models are then combined to produce the overall model output, which can be used to measure model performance.

Boosting is a technique designed to transform a set of so-called *weak learners* into a single *strong learner*. In plain English, think of a weak learner as a model that predicts only slightly better than random guessing, and a strong learner as a model that can predict to certain degree of accuracy better than random guessing.

While complicated, boosting basically works by iteratively creating weak models and adding them to the single strong learner. While this process happens, model accuracy is tested and then weightings are applied so that future learners focus on improving model performance for cases that were previously not well predicted.

Another very popular ensemble method is known as random forests. Random forests are essentially the combination of decision trees and bagging.

Kaggle is arguably the world’s most prestigious data science competition platform, and features competitions that are created and sponsored by most of the notable Silicon Valley tech companies, as well as by other very well-known corporations. Ensemble methods such as random forests and boosting have enjoyed very high success rates in winning these competitions.

Model Validation and Resampling Methods

Model validation is a very important part of the machine learning process. Validation methods consist of creating models and testing them on a validation dataset.

Resulting validation-set error provides an estimate of the test error and is typically assessed using *mean squared error* (*MSE*) in the case of a quantitative response, and misclassification rate in the case of a qualitative (discrete) response.

Many validation techniques are categorized as *resampling methods*, which involve refitting models to different samples formed from a set of training data.

Probably the most popular and noteworthy technique is called cross-validation. The key idea of cross-validation is that the model's accuracy on the training set is optimistic, and that a better estimate comes from the model's accuracy on the test set. The idea then is to estimate the test set accuracy while in the model training stage.

The process involves repeated splitting of the data into different training and test sets, building the model on the training set, and then evaluating it on the test set, and finally repeating and averaging the estimated errors.

In addition to model validation and helping to prevent overfitting, cross-validation can be used for feature selection, model selection, model parameter tuning, and comparing different predictors.

A popular special case of cross-validation is known as k-fold cross-validation. This technique involves selecting a number k, which represents the number of partitions of equal size that the original data is divided into. Once divided, a single partition is designated as a validation dataset (i.e., for testing the model), and the remaining k-1 data partitions are used as training data.

Note that typically the larger the chosen k, the less bias, but more variance, and vice versa. In the case of cross-validation, random sampling is done *without replacement*.

There is another technique that involves random sampling with replacement that is known as the *bootstrap*. The bootstrap technique tends to underestimate the error more than cross-validation.

Another special case is when k=n, i.e., when k equals the number of observations. In this case, the technique is known as *leave-one-out cross-validation* (*LOOCV*).

Residuals and Classification Results

Before digging deeper into model performance and error types, we must first discuss the concept of *residuals* and *errors* for regression, *positive* and *negative* classifications for classification problems, and *in-sample* versus *out-of-sample* measurements.

Any reference to models, metrics, or errors computed with respect to the data used to train, validate, or tune a predictive model (i.e., data you have) is called *in-sample*. Conversely, reference to test data metrics and errors, or new data in general is called *out-of-sample* (i.e., data you don't have).

Recall that regression involves predicting a continuous valued output (response) based on some set of input variables (features/predictors). The difference between the model's predicted response value and the actual observed response value from the *in-sample* data is called the *residual* for each point, and *residuals* refers collectively to all of the differences between all predicted and actual values. Each out-of-sample (new/test data) difference is called a *prediction error* instead of residual.

For the classification case, and for simplicity, we will only discuss binary classification (two classes). Prior to performing classification on data observations, one must define what is a positive classification and what is a negative classification. In the case of *spam* or *ham* (i.e., *not* spam), spam may be the positive designation and ham is the negative.

If a model predicts an incoming email as being spam, and it really is spam, then that's considered a true positive. Positive since the model predicted spam (the positive class), and true because the actual class matched the prediction. Conversely, if an incoming email is labeled spam when it's actually not spam, it is considered a false positive.

Given this, we can see that the results of a classification model on new data can fall into four potential buckets. These include: true positives, false positives (*type 1 error*), true negatives, and false negatives (*type 2 error*). In all four cases, true or false refers to whether the actual class matched the predicted class, and positive or negative refers to which classification was assigned to an observation by the model.

Note that *false* is synonymous with *error* in this case since the model failed to predict correctly.

Model Performance Overview

Now that we've covered residuals and classification result types, we will begin the discussion of model performance metrics that are based on these concepts.

Here is a non-exhaustive list of model evaluation methods, visualizations, and performance metrics that are used in machine learning and predictive analytics. They are categorized by their most common use case, but some may apply to more than one category (e.g., accuracy).

In addition to model evaluation, many of these can also be used for model comparison, selection, and tuning. Many of these are very powerful when combined with the cross-validation technique described earlier in this series.

- Regression performance
  - R^2 and adjusted R^2 (aka *explained variance*)
  - Mean squared error (MSE), or root mean squared error (RMSE)
  - Mean error, or mean absolute error
  - Median error, or median absolute error
- Classification performance
  - Confusion matrix
  - Precision
  - Recall (aka sensitivity)
  - Specificity
  - Accuracy
  - Lift
  - Area under the ROC curve (AUC)
  - F-score
  - Log-loss
  - Average precision
  - Precision/recall break-even point
  - Root mean squared error (RMSE)
  - Mean cross entropy
  - Probability calibration
- Bias variance tradeoff and model complexity
  - Validation curve
  - Learning curve
  - Residual sum of squares

    - Goodness-of-fit metrics
- Model validation and selection
    - Mallow's C_p
    - Akaike information criterion (AIC)
    - Bayesian information criterion (BIC)

Performance metrics should be chosen based on the problem domain, project goals, and the business objectives. Unfortunately there isn't a one-size-fits-all approach, and often there are tradeoffs to consider.

While a discussion of all of these methods and metrics is out of scope for this series, we will cover a few key ones next.

Model Performance Evaluation Metrics

Regression

There are many metrics for determining model performance for regression problems, but the most commonly used metric is known as the *mean square error* (*MSE*), or variation called the *root mean square error* (*RMSE*), which is calculated by taking the square root of the mean squared error. The root mean square error is typically preferred since taking the square root changes the units of the error measurement to be the same and proportional to the response variable's units.

The error in this case is the difference in value between a given model prediction and its actual value for an out-of-sample observation. The mean squared error is therefore the average of all of the squared errors across all new observations, which is the same as adding all of the squared errors (*sum of squares*) and dividing by the number of observations.

In addition to being used as a stand-alone performance metric, mean squared error (or RMSE) can also be used for model selection, controlling model complexity, and model tuning. Often many models are created and evaluated (e.g., cross-validation), and then MSE (or similar metric) is plotted on the y-axis, with the tuning or validation parameter given on the x-axis.

The tuning or validation parameter is changed in each model creation and evaluation step, and the plot described above can help determine the ideal tuning parameter value. The number of predictors is a great example of a potential tuning parameter in this case.

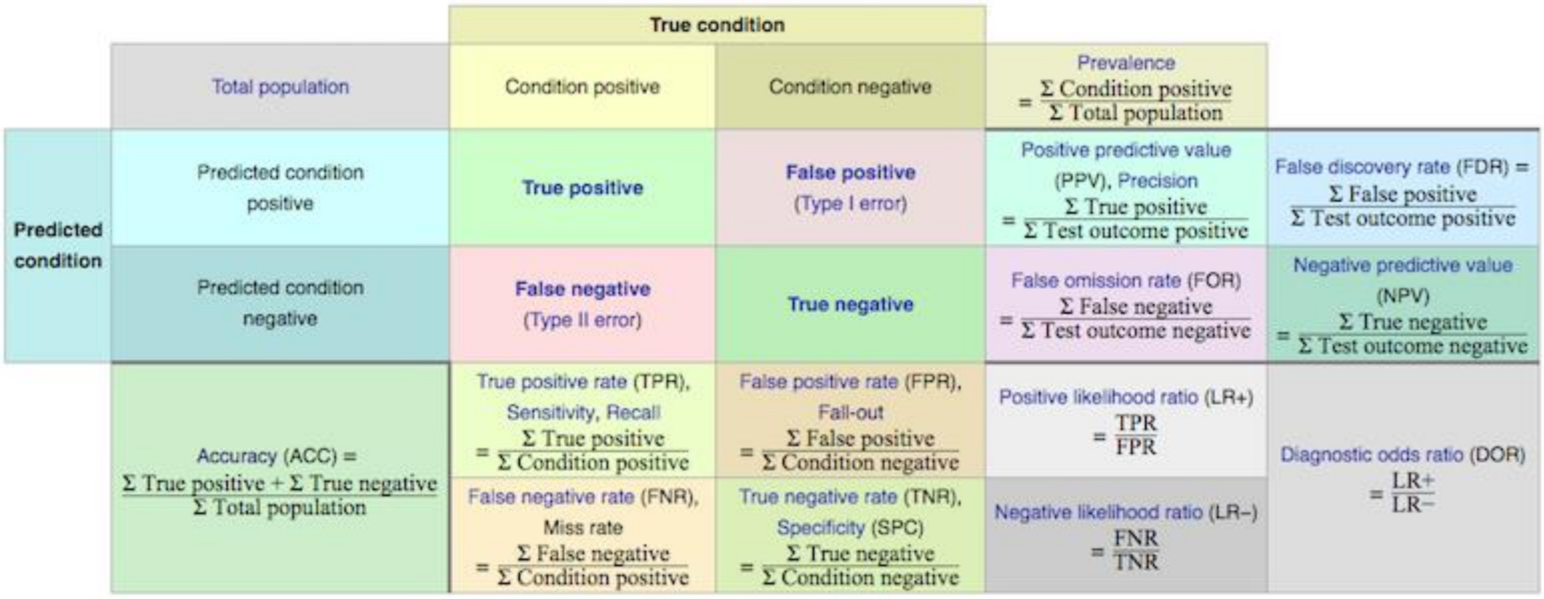
Before moving on to classification, it is worth mentioning R^2 briefly. R^2 is often thought of as a measure of model performance, but it's actually not. R^2 is a measure of the amount of variance explained by the model, and is given as a number between 0 and 1. A value of 1 means the model explains all of the data perfectly, but when computed with training data is more of an indication of potential overfitting than high predictive performance.

As discussed earlier, the more complex the model, the more the model *tends* to fit the data better and potentially overfit, or contribute to additional model *variance*. Given this, adjusted R^2 is a more robust and reliable metric in that it adjusts for any increases in model complexity (e.g., adding more predictors), so that one can better gauge underlying model improvement in lieu of the increased complexity.

Classification

Recall the different results from a binary classifier, which are true positives, true negatives, false positives, and false negatives. These are often shown in a *confusion matrix*. Here is a very generalized and comprehensive example of one from Wikipedia, and note that the graphic is shown with concepts and metrics, and not actual data.

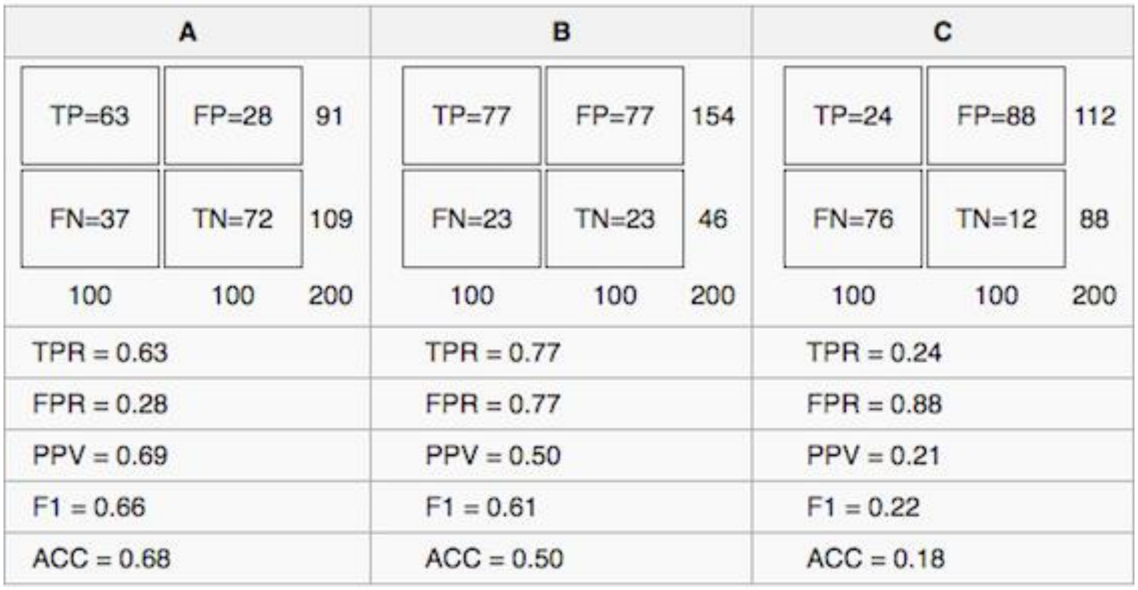
.



		True condition			
	Total population	Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive** (Type I error)	Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$	False discovery rate (FDR) $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Test outcome positive}}$
	Predicted condition negative	**False negative** (Type II error)	**True negative**	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$	Negative predictive value (NPV) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$
	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	True positive rate (TPR), Sensitivity, Recall $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$	

And here is an example from Wikipedia with the values filled in for different classifier models evaluated against 200 observations. Note the calculation and variation of the metrics across the different models.

.



A			B			C		
TP=63	FP=28	91	TP=77	FP=77	154	TP=24	FP=88	112
FN=37	TN=72	109	FN=23	TN=23	46	FN=76	TN=12	88
100	100	200	100	100	200	100	100	200
TPR = 0.63			TPR = 0.77			TPR = 0.24		
FPR = 0.28			FPR = 0.77			FPR = 0.88		
PPV = 0.69			PPV = 0.50			PPV = 0.21		
F1 = 0.66			F1 = 0.61			F1 = 0.22		
ACC = 0.68			ACC = 0.50			ACC = 0.18		

A confusion matrix is conceptually the basis of many classification performance metrics as shown. We will discuss a few of the more popular ones associated with machine learning here. *Accuracy* is a key measure of performance, and is more specifically the rate at which the model is able to predict the correct value (classification or regression) for a given data point or observation. In other words, accuracy is the proportion of correct predictions out of all predictions made.

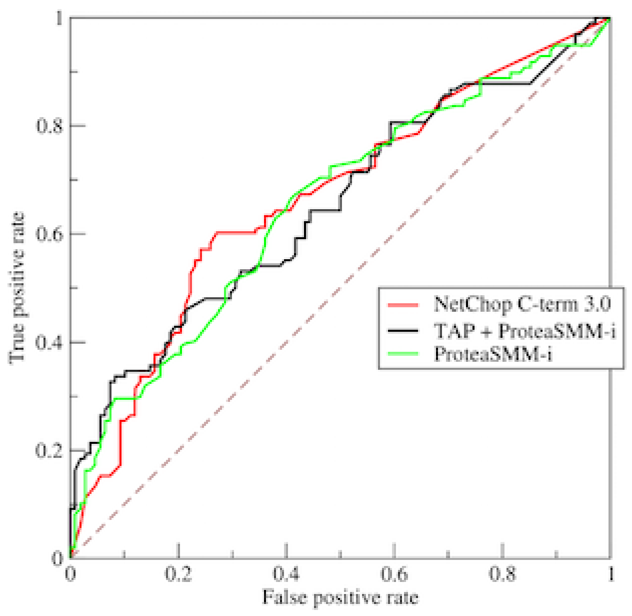
The other two metrics from the confusion matrix worth discussing are *precision* and *recall*. *Precision*(*positive predictive value*) is the ratio of true positives to the total amount of positive predictions made (i.e., true or false). Said another way, precision measures the proportion of accurate positive predictions out of all positive predictions made.

Recall on the other hand, or *true positive rate*, is the ratio of true positives to the total amount of actual positives, whether predicted correctly or not. So in other words, recall measures the proportion of accurate positive predictions out of all actual positive observations.

A metric that is associated with precision and recall is called the *F-score* (also called F_1 score), which combines them mathematically, and somewhat like a weighted average, in order to produce a single measure of performance based on the simultaneous values of both. Its values range from 0 (worst) to 1 (best).

Another important concept to know about is the *receiver operating characteristic*, which when plotted, results in what's known as an *ROC curve* (shown below, image courtesy of BOR at the English language Wikipedia).

An ROC curve is a two-dimensional plot of *sensitivity* (recall, or true positive rate) vs *specificity* (false positive rate). The area under the curve is referred to as the *AUC*, and is a numeric metric used to represent the quality and performance of the classifier (model).



An AUC of 0.5 is essentially the same as random guessing without a model, whereas an AUC of 1.0 is considered a perfect classifier. Generally, the higher the AUC value the better, and an AUC above 0.8 is considered quite good.

The higher the AUC value, the closer the curve gets to the upper left corner of the plot. One can easily see from the ROC curves then that the goal is to find and tune a model that maximizes the true positive rate, while simultaneously minimizing the false positive rate. Said another way, the goal as shown by the ROC curve is to correctly predict as many of the actual positives as possible, while also predicting as many of the actual negatives as possible, and therefore minimize errors (incorrect classifications) for both.

As mentioned previously in this series, model performance can be measured in many ways, and the method used should be chosen based on project goals, business domain considerations, and so on.

It is also worth noting that according to many experts, different performance metrics are thought to be biased for varying reasons. Given the breadth and complexity of this topic, the reader is encouraged to refer to external resources for further information on performance evaluation and the tradeoffs involved.

Error Analysis and Tradeoffs

There are multiple types of *errors* associated with machine learning and predictive analytics. The primary types are *in-sample* and *out-of-sample* errors. In-sample errors (aka *resubstitution errors*) are the error rate found from the training data, i.e., the data used to build predictive models.

Out-of-sample errors (aka *generalization errors*) are the error rates found on a new data set, and are the most important since they represent the potential performance of a given predictive model on new and unseen data.

In-sample error rates may be very low and seem to be indicative of a high-performing model, but one must be careful, as this may be due to overfitting as mentioned, which would result in a model that is unable to generalize well to new data.

Training and validation data is used to build, validate, and tune a model, but test data is used to evaluate model performance and generalization capability. One very important point to note is that prediction performance and error analysis should only be done on test data, when evaluating a model for use on non-training or new data (out-of-sample).

Generally speaking, model performance on training data tends to be optimistic, and therefore data errors will be less than those involving test data. There are tradeoffs between the types of errors that a machine learning practitioner must consider and often choose to accept.

For binary classification problems, there are two primary types of errors. *Type 1* errors (false positives) and *Type 2* errors (false negatives). It's often possible through model selection and tuning to increase one while decreasing the other, and often one must choose which error type is more acceptable. This can be a major tradeoff consideration depending on the situation.

A typical example of this tradeoff dilemma involves cancer diagnosis, where the positive diagnosis of having cancer is based on some test. In this case, a false positive means that someone is told that have have cancer when they do not. Conversely, the false negative case is when someone is told that they do not have cancer when they actually do.

If no model is perfect, then in the example above, which is the more acceptable error type? In other words, of which one can we accept to a greater degree?

Telling someone they have cancer when they don't can result in tremendous emotional distress, stress, additional tests and medical costs, and so on. On the other hand, failing to detect cancer in someone that actually has it can mean the difference between life and death.

In the spam or ham case, neither error type is nearly as serious as the cancer case, but typically email vendors err slightly more on the side of letting some spam get into your inbox as opposed to you missing a very important email because the spam classifier is too aggressive.

Unsupervised Learning

Recall that unsupervised learning involves learning from data, but without the goal of prediction. This is because the data is either not given with a target *response* variable (*label*), or one chooses not to designate a response. It can also be used as a pre-processing step for supervised learning.

In the unsupervised case, the goal is to discover patterns, deep insights, understand variation, find unknown subgroups (amongst the variables or observations), and so on in the data. Unsupervised learning can be quite subjective compared to supervised learning.

The two most commonly used techniques in unsupervised learning are *principal component analysis*(*PCA*) and *clustering*. PCA is one approach to learning what is called a *latent variable model*, and is a particular version of a *blind signal separation* technique. Other notable latent variable modeling approaches include *expectation-maximization algorithm (EM)* and *Method of moments*.

PCA

PCA produces a low-dimensional representation of a dataset by finding a sequence of linear combinations of the variables that have maximal variance, and are mutually uncorrelated. Another way to describe PCA is that it is a transformation of possibly correlated variables into a set of linearly uncorrelated variables known as principal components.

Each of the *components* are mathematically determined and ordered by the amount of variability or variance that each is able to explain from the data. Given that, the first principal component accounts for the largest amount of variance, the second principal component the next largest, and so on.

Each *component* is also *orthogonal* to all others, which is just a fancy way of saying that they're perpendicular to each other. Think of the X and Y axis' in a two dimensional plot. Both axis are perpendicular to each other, and are therefore *orthogonal*. While not easy to visualize, think of having many principal components as being many axis that are perpendicular to each other.

While much of the above description of principal component analysis may be a bit technical sounding, it is actually a relatively simple concept from a high level. Think of having a bunch of data in any amount of dimensions, although you may want to picture two or three dimensions for ease of understanding.

Each principal component can be thought of as an axis of an ellipse that is being built (think cloud) to contain the data (aka fit to the data), like a net catching butterflies. The first few principal components should be able to explain (capture) most of the data, with the addition of more principal components eventually leading to diminishing returns.

One of the tricks of PCA is knowing how many components are needed to summarize the data, which involves estimating when most of the variance is explained by a given number of components. Another consideration is that PCA is sensitive to feature scaling, which was discussed earlier in this series.

PCA is also used for *exploratory data analysis* and *data visualization*. Exploratory data analysis involves summarizing a dataset through specific types of analysis, including data visualization, and is often an initial step in analytics that leads to predictive modeling, data mining, and so on.

Further discussion of PCA and similar techniques is out of scope of this series, but the reader is encouraged to refer to external sources for more information.

Clustering

Clustering refers to a set of techniques and algorithms used to find *clusters* (subgroups) in a dataset, and involves partitioning the data into groups of similar observations. The concept of 'similar observations' is a bit relative and subjective, but it essentially means that the data points in a given group are more similar to each other than they are to data points in a different group.

Similarity between observations is a domain specific problem and must be addressed accordingly. A clustering example involving the NFL's Chicago Bears (go Bears!) was given in the first article of this series.

Clustering is not a technique limited only to machine learning. It is a widely used technique in data mining, statistical analysis, pattern recognition, image analysis, and so on. Given the subjective and unsupervised nature of clustering, often data preprocessing, model/algorithm selection, and model tuning are the best tools to use to achieve the desired results and/or solution to a problem.

There are many types of clustering algorithms and models, which all use their own technique of dividing the data into a certain number of groups of similar data. Due to the significant difference in these approaches, the results can be largely affected, and therefore one must understand these different algorithms to some extent to choose the most applicable approach to use.

K-means and hierarchical clustering are two widely used unsupervised clustering techniques. The difference is that for k-means, a predetermined number of clusters (k) is used to partition the observations, whereas the number of clusters in hierarchical clustering is not known in advance.

Hierarchical clustering helps address the potential disadvantage of having to know or pre-determine k in the case of k-means. There are two primary types of hierarchical clustering, which include bottom-up and agglomerative.

Here is a visualization, courtesy of Wikipedia, of the results of running the k-means clustering algorithm on a set of data with k equal to three. Note the lines, which represent the boundaries between the groups of data.

There are two types of clustering, which define the degree of grouping or containment of data. The first is called *hard clustering*, where every data point belongs to only one cluster and not the others. *Soft clustering*, or *fuzzy clustering* on the other hand refers to the case where a data point belongs to a cluster to a certain degree, or is assigned a likelihood (probability) of belonging to a certain cluster.

Method comparison and general considerations

What is the difference then between PCA and clustering? As mentioned, PCA looks for a low-dimensional representation of the observations that explains a good fraction of the variance, while clustering looks for homogeneous subgroups among the observations.

An interesting point to note is that in the absence of a target response, there is no way to evaluate solution performance or errors as one does in the supervised case. In other words, there is no objective way to determine if you’ve found a solution. This is a significant differentiator between supervised and unsupervised learning methods.

Predictive Analytics, Artificial Intelligence, and Data Mining

Machine learning is often interchanged with terms like predictive analytics, artificial intelligence, data mining, and so on. While machine learning is certainly related to these fields, there are some notable differences.

Predictive analytics is a subcategory of a broader field known as analytics in general. Analytics is usually broken into three sub-categories: descriptive, predictive, and prescriptive.

Descriptive analytics involves analytics applied to understanding and describing data. Predictive analytics deals with modeling, and making predictions or assigning classifications from data observations. Prescriptive analytics deals with making data-driven, actionable recommendations or decisions.

Artificial intelligence (AI) is a super exciting field, and machine learning is essentially a sub-field of AI due to the automated nature of the learning algorithms involved. According to Wikipedia, AI has been defined as *the science and engineering of making intelligent machines*, but also as *the study and design of intelligent agents*, where an *intelligent agent* is a system that perceives its environment and takes actions that maximize its chances of success

Statistical learning is becoming popularized due to Stanford’s related online course and its associated books: *An Introduction to Statistical Learning*, and *The Elements of Statistical Learning*.

Machine learning arose as a subfield of artificial intelligence, statistical learning arose as a subfield of statistics. Both fields are very similar, overlap in many ways, and the distinction is becoming less clear over time. They differ in that machine learning has a greater emphasis on prediction accuracy and large scale applications, whereas statistical learning emphasizes models and their related interpretability, precision, and uncertainty.

Lastly, data mining is a field that’s also often confused with machine learning. Data mining leverages machine learning algorithms and techniques, but also spans many other fields such as data science, AI, statistics, and so on.

The overall goal of the data mining process is to extract patterns and knowledge from a data set, and transform it into an understandable structure for further use. Data mining often deals with large amounts of data, or *big data*.

Machine Learning in Practice

As discussed throughout this series, machine learning can be used to create predictive models, assign classifications, make recommendations, and find patterns and insights in an unlabeled dataset. All of these tasks can be done without requiring explicit programming.

Machine learning has been successfully used in the following non-exhaustive example applications:

- Spam filtering
- Optical character recognition (OCR)
- Search engines
- Computer vision
- Recommendation engines, such as those used by Netflix and Amazon

- Classifying DNA sequences
- Detecting fraud, e.g., credit card and internet
- Medical diagnosis
- Natural language processing
- Speech and handwriting recognition
- Economics and finance
- Virtually anything else you can think of that involves data

In order to apply machine learning to solve a given problem, the following steps (or a variation) should to be taken, and should use machine learning elements discussed throughout this series.

1. Define the problem to be solved and the project's objective. Ask lots of questions along the way!
2. Determine the type of problem and type of solution required.
3. Collect and prepare the data.
4. Create, validate, tune, test, assess, and improve your model and/or solution. This process should be driven by a combination of technical (stats, math, programming), domain, and business expertise.
5. Discover any other insights and patterns as applicable.
6. Deploy your solution for real-world use.
7. Report on and/or present results.

If you encounter a situation where you or your company can benefit from a machine learning-based solution, simply approach it using these steps and see what you come up with. You may very well wind up with a super powerful and scalable solution!

Additional Reading

Artificial intelligence (AI), deep learning, and neural networks

Artificial intelligence (AI), deep learning, and neural networks represent incredibly exciting and powerful machine learning-based techniques used to solve many real-world problems.

While human-like deductive reasoning, inference, and decision-making by a computer is still a long time away, there have been remarkable gains in the application of AI techniques and associated algorithms.

The primary motivation and driving force for these areas of study, and for developing these techniques further, is that the solutions required to solve certain problems are incredibly complicated, not well understood, nor easy to determine manually.

Increasingly, we rely on these techniques and machine learning to solve these problems for us, without requiring explicit programming instructions. This is critical for two reasons. The first is that we likely wouldn't be able, or at least know how to write the programs required to model and solve many problems that AI techniques are able to solve. Second, even if we did know how to write the programs, they would be inordinately complex and nearly impossible to get right.

Luckily for us, machine learning and AI algorithms, along with properly selected and prepared training data, are able to do this for us.

Artificial Intelligence Overview

In order to define AI, we must first define the concept of *intelligence* in general. A paraphrased definition based on Wikipedia is:

Intelligence can be generally described as the ability to perceive information, and retain it as knowledge to be applied towards adaptive behaviors within an environment or context.

While there are many different definitions of intelligence, they all essentially involve learning, understanding, and the application of the knowledge learned to achieve one or more goals.

It's therefore a natural extension to say that AI can be described as intelligence exhibited by machines. So what does that mean exactly, when is it useful, and how does it work?

A familiar instance of an AI solution includes *IBM's Watson*, which was made famous by beating the two greatest *Jeopardy* champions in history, and is now being used as a *question answering computing system* for commercial applications. *Apple's Siri* and*Amazon's Alexa* are similar examples as well.

In addition to speech recognition and natural language (processing, generation, and understanding) applications, AI is also used for other recognition tasks (pattern, text, audio, image, video, facial, …), autonomous vehicles, medical diagnoses, gaming, search engines, spam filtering, crime fighting, marketing, robotics, remote sensing, computer vision, transportation, music recognition, classification, and so on.

Something worth mentioning is a concept known as the *AI effect*. This describes the case where once an AI application has become somewhat mainstream, it's no longer considered by many as AI. It happens because people's tendency is to no longer think of the solution as involving real intelligence, and only being a application of normal computing.

This despite the fact that these applications still fit the definition of AI regardless of widespread usage. The key takeaway here is that today's AI is not necessarily tomorrow's AI, at least not in some people's minds anyway.

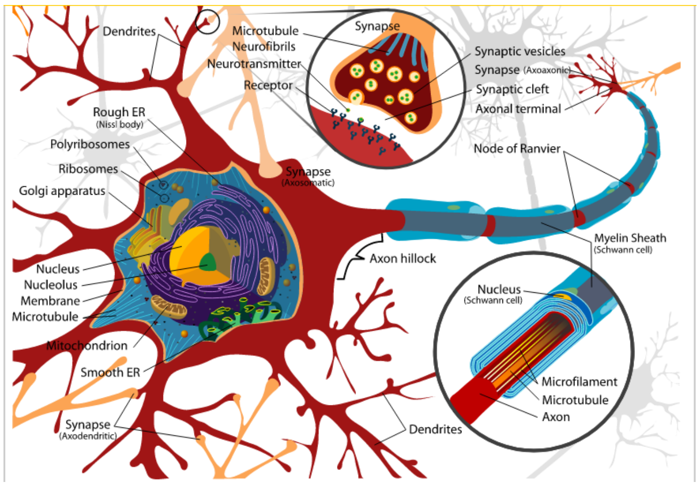
There are many different goals of AI as mentioned, with different techniques used for each. The primary topics of this article are artificial neural networks and an advanced version known as deep learning.

Biological Neural Networks Overview

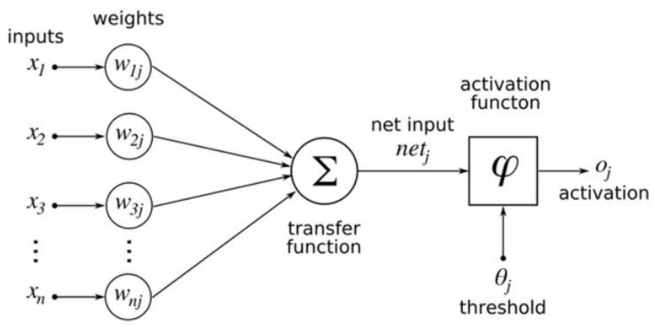
The human brain is exceptionally complex and quite literally the most powerful computing machine known.

The inner-workings of the human brain are often modeled around the concept of*neurons* and the networks of neurons known as *biological neural networks*. According to Wikipedia, it's estimated that the human brain contains roughly 100 billion neurons, which are connected along pathways throughout these networks.

At a very high level, neurons interact and communicate with one another through an interface consisting of axon terminals that are connected to dendrites across a gap (synapse) as shown here.



In plain english, a single neuron will pass a message to another neuron across this interface if the sum of *weighted input signals* from one or more neurons (*summation*) into it is great enough (exceeds a *threshold*) to cause the message transmission. This is called *activation* when the threshold is exceeded and the message is passed along to the next neuron.



The *summation* process can be mathematically complex. Each neuron's input signal is actually a *weighted combination* of potentially many input signals, and the weighting of each input means that that input can have a different influence on any subsequent calculations, and ultimately on the final output of the entire network.

In addition, each neuron applies a function or *transformation* to the weighted inputs, which means that the combined weighted input signal is transformed mathematically prior to evaluating if the *activation threshold* has been exceeded. This combination of weighted input signals and the functions applied are typically either linear or nonlinear.

These input signals can originate in many ways, with our senses being some of the most important, as well as ingestion of gases (breathing), liquids (drinking), and solids (eating) for example. A single neuron may receive hundreds of thousands of input signals at once that undergo the summation process to determine if the message gets passed along, and ultimately causes the brain to instruct actions, memory recollection, and so on.

The 'thinking' or processing that our brain carries out, and the subsequent instructions given to our muscles, organs, and body are the result of these neural networks in action. In addition, the brain's neural networks continuously change and update themselves in many ways, including modifications to the amount of weighting applied between neurons. This happens as a direct result of learning and experience.

Given this, it’s a natural assumption that for a computing machine to replicate the brain’s functionality and capabilities, including being ‘intelligent’, it must successfully implement a computer-based or artificial version of this network of neurons.

This is the genesis of the advanced statistical technique and term known as *artificial neural networks*.

Artificial Neural Networks Overview

Artificial neural networks (*ANNs*) are statistical models directly inspired by, and partially modeled on biological neural networks. They are capable of modeling and processing nonlinear relationships between inputs and outputs in parallel. The related algorithms are part of the broader field of machine learning, and can be used in many applications as discussed.

Artificial neural networks are characterized by containing *adaptive weights* along paths between neurons that can be tuned by a *learning algorithm* that *learns* from observed data in order to improve the model. In addition to the learning algorithm itself, one must choose an appropriate *cost function*.

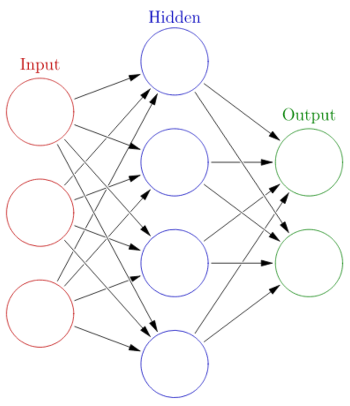
The cost function is what’s used to *learn* the optimal solution to the problem being solved. This involves determining the best values for all of the tunable model parameters, with neuron path adaptive weights being the primary target, along with algorithm tuning parameters such as the *learning rate*. It’s usually done through*optimization* techniques such as *gradient descent* or *stochastic gradient descent*.

These optimization techniques basically try to make the ANN solution be as close as possible to the optimal solution, which when successful means that the ANN is able to solve the intended problem with high performance.

Architecturally, an artificial neural network is modeled using layers of *artificial neurons*, or computational units able to receive input and apply an activation function along with a threshold to determine if messages are passed along.

In a simple model, the first layer is the *input* layer, followed by one *hidden* layer, and lastly by an *output* layer. Each layer can contain one or more neurons.

Models can become increasingly complex, and with increased abstraction and problem solving capabilities by increasing the number of *hidden layers*, the number of neurons in any given layer, and/or the number of paths between neurons. Note that an increased chance of *overfitting* can also occur with increased model complexity.



Model architecture and tuning are therefore major components of ANN techniques, in addition to the actual learning algorithms themselves. All of these characteristics of an ANN can have significant impact on the performance of the model.

Additionally, models are characterized and tunable by the *activation function* used to convert a neuron's weighted input to its output activation. There are many different types of transformations that can be used as the activation function, and a discussion of them is out of scope for this article.

The abstraction of the output as a result of the transformations of input data through neurons and layers is a form of *distributed representation*, as contrasted with*local representation*. The meaning represented by a single artificial neuron for example is a form of local representation. The meaning of the entire network however, is a form of distributed representation due to the many transformations across neurons and layers.

One thing worth noting is that while ANNs are extremely powerful, they can also be very complex and are considered *black box* algorithms, which means that their inner-workings are very difficult to understand and explain. Choosing whether to employ ANNs to solve problems should therefore be chosen with that in mind.

Deep Learning Introduction

Deep learning, while sounding flashy, is really just a term to describe certain types of neural networks and related algorithms that consume often very *raw* input data. They process this data through many layers of nonlinear transformations of the input data in order to calculate a target output.
Unsupervised *feature extraction* is also an area where deep learning excels. Feature extraction is when an algorithm is able to automatically derive or construct meaningful features of the data to be used for further learning, generalization, and understanding. The burden is traditionally on the data scientist or programmer to carry out the feature extraction process in most other machine learning approaches, along with feature selection and engineering.

Feature extraction usually involves some amount dimensionality reduction as well, which is reducing the amount of input features and data required to generate meaningful results. This has many benefits, which include simplification, computational and memory power reduction, and so on.

More generally, deep learning falls under the group of techniques known as *feature learning* or *representation learning*. As discussed so far, feature extraction is used to 'learn' which features to focus on and use in machine learning solutions. The machine learning algorithms themselves 'learn' the optimal parameters to create the best performing model.

Paraphrasing Wikipedia, *feature learning* algorithms allow a machine to both learn for a specific task using a well-suited set of features, and also learn the features themselves. In other words, these algorithms *learn how to learn*!
Deep learning has been used successfully in many applications, and is considered to be one of the most cutting-edge machine learning and AI techniques at the time of this writing. The associated algorithms are often used for *supervised*, *unsupervised*, and *semi-supervised* learning problems.

For neural network-based deep learning models, the number of layers are greater than in so-called *shallow learning* algorithms. Shallow algorithms tend to be less complex and require more up-front knowledge of optimal features to use, which typically involves feature selection and engineering.
In contrast, deep learning algorithms rely more on optimal model selection and optimization through model tuning. They are more well suited to solve problems where prior knowledge of features is less desired or necessary, and where labeled data is unavailable or not required for the primary use case.

In addition to statistical techniques, neural networks and deep learning leverage concepts and techniques from signal processing as well, including nonlinear processing and/or transformations.

You may recall that a nonlinear function is one that is not characterized simply by a straight line. It therefore requires more than just a slope to model the relationship between the input, or independent variable, and the output, or dependent variable. Nonlinear functions can include polynomial, logarithmic, and exponential terms, as well as any other transformation that isn't linear.

Many phenomena observed in the physical universe are actually best modeled with nonlinear transformations. This is true as well for transformations between inputs and the target output in machine learning and AI solutions.

A Deeper Dive into Deep Learning

As mentioned, input data is transformed throughout the layers of a deep learning neural network by artificial neurons or processing units. The chain of transformations that occur from input to output is known as the *credit assignment path*, or *CAP*.

The CAP value is a proxy for the measurement or concept of 'depth' in a deep learning model architecture. According to Wikipedia, most researchers in the field agree that deep learning has multiple nonlinear layers with a CAP greater than two, and some consider a CAP greater than ten to be *very deep learning*.

While a detailed discussion of the many different deep-learning model architectures and learning algorithms is beyond the scope of this article, some of the more notable ones include:

- Feed-forward neural networks
- Recurrent neural network
- Multi-layer perceptrons (MLP)
- Convolutional neural networks
- Recursive neural networks
- Deep belief networks
- Convolutional deep belief networks
- Self-Organizing Maps
- Deep Boltzmann machines
- Stacked de-noising auto-encoders

It's worth pointing out that due to the relative increase in complexity, deep learning and neural network algorithms can be prone to overfitting. In addition, increased model and algorithmic complexity can result in very significant computational resource and time requirements.

It's also important to consider that solutions may represent *local minima* as opposed to a global optimal solution. This is due to the complex nature of these models when combined with optimization techniques such as gradient descent.

Given all of this, proper care must be taken when leveraging artificial intelligence algorithms to solve problems, including the selection, implementation, and performance assessment of algorithms themselves. While out of scope for this article, the field of machine learning includes many techniques that can help with these areas.

Cognitive Computing

The goal of cognitive computing is to simulate human thought processes in a computerized model. Using self-learning algorithms that use data mining, pattern recognition and natural language processing, the computer can mimic the way the human brain works.

While computers have been faster at calculations and processing than humans for decades, they haven’t been able to accomplish tasks that humans take for granted as simple, like understanding natural language, or recognizing unique objects in an image.

Some people say that cognitive computing represents the third era of computing: we went from computers that could tabulate sums (1900s) to programmable systems (1950s), and now to cognitive systems.

These cognitive systems, most notably IBM Watson, rely on deep learning algorithms and neural networks to process information by comparing it to a teaching set of data. The more data the system is exposed to, the more it learns, and the more accurate it becomes over time, and the neural network is a complex “tree” of decisions the computer can make to arrive at an answer.

What can cognitive computing do?

For example, according to a TED Talk video from IBM, Watson could eventually be applied in a healthcare setting to help collate the span of knowledge around a condition, including patient history, journal articles, best practices, diagnostic tools, etc., analyze that vast quantity of information, and provide a recommendation.

The doctor is then able to look at evidence-based treatment options based on a large number of factors including the individual patient’s presentation and history, to hopefully make better treatment decisions.

In other words, the goal (at this point) is not to replace the doctor, but expand the doctor’s capabilities by processing the humongous amount of data available that no human could reasonably process and retain, and provide a summary and potential application.

This sort of process could be done for any field in which large quantities of complex data need to be processed and analyzed to solve problems, including finance, law, and education.

These systems will also be applied in other areas of business including consumer behavior analysis, personal shopping bots, customer support bots, travel agents, tutors, security, and diagnostics. Hilton Hotels recently debuted the first concierge robot, Connie, which can answer questions about the hotel, local attractions, and restaurants posed to it in natural language.

The personal digital assistants we have on our phones and computers now (Siri and Google among others) are not true cognitive systems; they have a pre-programmed set of responses and can only respond to a preset number of requests. But the time is coming in the near future when we will be able to address our phones, our computers, our cars, or our smart houses and get a real, thoughtful response rather than a pre-programmed one.

As computers become more able to think like human beings, they will also expand our capabilities and knowledge. Just as the heroes of science fiction movies rely on their computers to make accurate predictions, gather data, and draw conclusions, so we will move into an era when computers can augment human knowledge and ingenuity in entirely new ways.