Machine Learning Notes

Prediction improvement tips

Improve Performance With Data

- **Get More Data.** Can you get more or better quality data? Modern nonlinear machine learning techniques like deep learning continue to improve in performance with more data.
- **Invent More Data**. If you can't get more data, can you generate new data? Perhaps you can augment or permute existing data or use a probabilistic model to generate new data.
- **Clean Your Data.** Can you improve the signal in your data? Perhaps there are missing or corrupt observations that can be fixed or removed, or outlier values outside of reasonable ranges that can be fixed or removed in order to lift the quality of your data.
- **Resample Data.** Can you resample data to change the size or distribution? Perhaps you can use a much smaller sample of data for your experiments to speed things up or over-sample or under-sample observations of a specific type to better represent them in your dataset.
- **Reframe Your Problem.** Can you change the type of prediction problem you are solving? Reframe your data as a regression, binary or multi-class classification, time series, anomaly detection, rating, recommender, etc. type problem.
- **Rescale Your Data.** Can you rescale numeric input variables? Normalization and standardization of input data can result in a lift in performance on algorithms that use weighted inputs or distance measures.
- **Transform Your Data.** Can you reshape your data distribution? Making input data more Gaussian or passing it through an exponential function may better expose features in the data to a learning algorithm.
- **Project Your Data.** Can you project your data into a lower dimensional space? You can use an unsupervised clustering or projection method to create an entirely new compressed representation of your dataset.
- **Feature Selection.** Are all input variables equally important? Use feature selection and feature importance methods to create new views of your data to explore with modeling algorithms.
- **Feature Engineering.** Can you create and add new data features? Perhaps there are attributes that can be decomposed into multiple new values (like categories, dates or strings) or attributes that can be aggregated to signify an event (like a count, binary or statistical summary).

Improve Performance With Algorithms

- **Resampling Method**. What resampling method is used to estimate skill on new data? Use a method and configuration that makes the best use of available data. The k-fold cross-validation method with a hold out validation dataset might be a best practice.
- **Evaluation Metric**. What metric is used to evaluate the skill of predictions? Use a metric that best captures the requirements of the problem and the domain. It probably isn't classification accuracy.
- **Baseline Performance**. What is the baseline performance for comparing algorithms? Use a random algorithm or a zero rule algorithm (predict mean or mode) to establish a baseline by which to rank all evaluated algorithms.
- **Spot-Check Linear Algorithms**. What linear algorithms work well? Linear methods are often more biased, are easy to understand and are fast to train. They are preferred if you can achieve good results. Evaluate a diverse suite of linear methods.
- **Spot-Check Nonlinear Algorithms**. What nonlinear algorithms work well? Nonlinear algorithms often require more data, have greater complexity but can achieve better performance. Evaluate a diverse suite of nonlinear methods.
- **Steal from Literature**. What algorithms are reported in the literature to work well on your problem? Perhaps you can get ideas of algorithm types or extensions of classical methods to explore on your problem.
- **Standard Configurations**. What are the standard configurations for the algorithms being evaluated? Each algorithm needs an opportunity to do well on your problem. This does not mean tune the parameters (yet) but it does mean to investigate how to configure each algorithm well and give it a fighting chance in the algorithm bake-off.

Improve Performance With Tuning

- **Diagnostics**. What do diagnostics tell you about your algorithm? Perhaps you can review learning curves to understand whether the method is over or underfitting the problem, and then correct. Different algorithms may offer different visualizations and diagnostics. Review what the algorithm is predicting right and wrong.
- **Try Intuition**. What does your gut tell you? If you fiddle with parameters for long enough and the feedback cycle is short, you can develop an intuition for how to configure an algorithm on a problem. Try this out and see if you can come up with new parameter configurations to try on your larger test harness.
- **Steal from Literature**. What parameters or parameter ranges are used in the literature? Evaluating the performance of standard parameters is a great place to start any tuning activity.
- **Random Search**. What parameters can use random search? Perhaps you can use random search of algorithm hyperparameters to expose configurations that you would never think to try.
- **Grid Search**. What parameters can use grid search? Perhaps there are grids of standard hyperparameter values that you can enumerate to find good configurations, then repeat the process with finer and finer grids.
- **Optimize**. What parameters can you optimize? Perhaps there are parameters like structure or learning rate than can be tuned using a direct search procedure (like pattern search) or stochastic optimization (like a genetic algorithm).
- **Alternate Implementations**. What other implementations of the algorithm are available? Perhaps an alternate implementation of the method can achieve better results on the same data. Each algorithm has a myriad of micro-decisions that must be made by the algorithm implementor. Some of these decisions may affect skill on your problem.
- **Algorithm Extensions**. What are common extensions to the algorithm? Perhaps you can lift performance by evaluating common or standard extensions to the method. This may require implementation work.
- **Algorithm Customizations**. What customizations can be made to the algorithm for your specific case? Perhaps there are modications that you can make to the algorithm for your data, from loss function, internal optimization methods to algorithm specific decisions.
- **Contact Experts**. What do algorithm experts recommend in your case? Write a short email summarizing your prediction problem and what you have tried to one or more expert academics on the algorithm. This may reveal leading edge work or academic work previously unknown to you with new or fresh ideas.

Improve Performance With Ensembles

- **Blend Model Predictions**. Can you combine the predictions from multiple models directly? Perhaps you could use the same or different algorithms to make multiple models. Take the mean or mode from the predictions of multiple well-performing models.
- **Blend Data Representations**. Can you combine predictions from models trained on different data representations? You may have many different projections of your problem which can be used to train well-performing algorithms, whose predictions can then be combined.
- **Blend Data Samples**. Can you combine models trained on different views of your data? Perhaps you can create multiple subsamples of your training data and train a well-performing algorithm, then combine predictions. This is called bootstrap aggregation or bagging and works best when the predictions from each model are skillful but in different ways (uncorrelated).
- **Correct Predictions**. Can you correct the predictions of well-performing models? Perhaps you can explicitly correct predictions or use a method like boosting to learn how to correct prediction errors.
- **Learn to Combine**. Can you use a new model to learn how to best combine the predictions from multiple well-performing models? This is called stacked generalization or stacking and often works well when the submodels are skillful but in different ways and the aggregator model is a simple linear weighting of the predictions. This process can be repeated multiple layers deep.

How to implement above four performance improvement tips

- Pick one group
    - Data.
    - Algorithms.
    - Tuning.
    - Ensembles.
- Pick one method from the group.
- Pick one thing to try of the chosen method.
- Compare the results, keep if there was an improvement.
- Repeat.

5-Step Systematic Process

- **Define the Problem**
    - **What is the problem?** Describe the problem informally and formally and list assumptions and similar problems.
    - **Why does the problem need to be solved?** List your motivation for solving the problem, the benefits a solution provides and how the solution will be used.
    - **How would I solve the problem?** Describe how the problem would be solved manually to flush domain knowledge.
- **Prepare Data** (data visualisation is very important before feeding it to the algorithm)
    - **Data Selection** Consider what data is available, what data is missing and what data can be removed.
        - What is the extent of the data you have available? For example through time, database tables, connected systems. Ensure you have a clear picture of everything that you can use.
        - What data is not available that you wish you had available? For example data that is not recorded or cannot be recorded. You may be able to derive or simulate this data.
        - What data don't you need to address the problem? Excluding data is almost always easier than including data. Note down which data you excluded and why.
    - **Data Preprocessing** Organize your selected data by formatting, cleaning and sampling from it.
        - **Formatting**: The data you have selected may not be in a format that is suitable for you to work with. The data may be in a relational database and you would like it in a flat file, or the data may be in a proprietary file format and you would like it in a relational database or a text file.
        - **Cleaning**: Cleaning data is the removal or fixing of missing data. There may be data instances that are incomplete and do not carry the data you believe you need to address the problem. These instances may need to be removed. Additionally, there may be sensitive information in some of the attributes and these attributes may need to be anonymized or removed from the data entirely.
        - **Sampling**: There may be far more selected data available than you need to work with. More data can result in much longer running times for algorithms and larger computational and memory requirements. You can take a smaller representative sample of the selected data that may be much faster for exploring and prototyping solutions before considering the whole dataset.
    - **Add Data Attributes** Advanced models can extract the relationships from complex attributes, although some models require those relationships to be spelled out plainly. Deriving new attributes from your training data to include in the modeling process can give you a boost in model performance.
        - **Dummy Attributes**: Categorical attributes can be converted into n-binary attributes, where n is the number of categories (or levels) that the attribute has. These denormalized or decomposed attributes are known as dummy attributes or dummy variables.
        - **Transformed Attribute**: A transformed variation of an attribute can be added to the dataset in order to allow a linear method to exploit possible linear and non-linear relationships between attributes. Simple transforms like log, square and square root can be used.
        - **Missing Data**: Attributes with missing data can have that missing data imputed using a reliable method, such as k-nearest neighbors.
    - **Remove Data Attributes** Some methods perform poorly with redundant or duplicate attributes. You can get a boost in model accuracy by removing attributes from your data.
        - **Projection**: Training data can be projected into lower dimensional spaces, but still characterize the inherent relationships in the data. A popular approach is Principal Component Analysis (PCA) where the principal components found by the method can be taken as a reduced set of input attributes.
        - **Spatial Sign**: A spatial sign projection of the data will transform data onto the surface of a multidimensional sphere. The results can be used to highlight the existence of outliers that can be modified or removed from the data.
        - **Correlated Attributes**: Some algorithms degrade in importance with the existence of highly correlated attributes. Pairwise attributes with high correlation can be identified and the most correlated attributes can be removed from the data.
    - **Data Transformation** Transform preprocessed data ready for machine learning by engineering features using scaling, attribute decomposition and attribute aggregation.

- **Centering**: Transform the data so that it has a mean of zero and a standard deviation of one. This is typically called data standardization.
- **Scaling**: The preprocessed data may contain attributes with a mixtures of scales for various quantities such as dollars, kilograms and sales volume. Many machine learning methods like data attributes to have the same scale such as between 0 and 1 for the smallest and largest value for a given feature. Consider any feature scaling you may need to perform.
- **Remove Skew**: Skewed data is data that has a distribution that is pushed to one side or the other (larger or smaller values) rather than being normally distributed. Some methods assume normally distributed data and can perform better if the skew is removed. Try replacing the attribute with the log, square root or inverse of the values.
- **Box-Cox**: A Box-Cox transform or family of transforms can be used to reliably adjust data to remove skew.
- **Binning**: Numeric data can be made discrete by grouping values into bins. This is typically called data discretization. This process can be performed manually, although is more reliable if performed systematically and automatically using a heuristic that makes sense in the domain.
- **Decomposition**: There may be features that represent a complex concept that may be more useful to a machine learning method when split into the constituent parts. An example is a date that may have day and time components that in turn could be split out further. Perhaps only the hour of day is relevant to the problem being solved. consider what feature decompositions you can perform.
- **Aggregation**: There may be features that can be aggregated into a single feature that would be more meaningful to the problem you are trying to solve. For example, there may be a data instances for each time a customer logged into a system that could be aggregated into a count for the number of logins allowing the additional instances to be discarded. Consider what type of feature aggregations could perform. For tabular data, this might include projection methods like Principal Component Analysis and unsupervised clustering methods. For image data, this might include line or edge detection.

- **Outlier Modeling** Remove outlier data from main dataset.
  - **Extreme Value Analysis**: Determine the statistical tails of the underlying distribution of the data. For example, statistical methods like the z-scores on univariate data.
    - Focus on univariate methods
    - Visualize the data using scatterplots, histograms and box and whisker plots and look for extreme values
    - Assume a distribution (Gaussian) and look for values more than 2 or 3 standard deviations from the mean or 1.5 times from the first or third quartile
    - Filter out outliers candidate from training dataset and assess your models performance
  - **Probabilistic and Statistical Models**: Determine unlikely instances from a probabilistic model of the data. For example, gaussian mixture models optimized using expectation-maximization.
  - **Linear Models**: Projection methods that model the data into lower dimensions using linear correlations. For example, principle component analysis and data with large residual errors may be outliers.
  - **Proximity-based Models**: Data instances that are isolated from the mass of the data as determined by cluster, density or nearest neighbor analysis.
    - Use clustering methods to identify the natural clusters in the data (such as the k-means algorithm)
    - Identify and mark the cluster centroids
    - Identify data instances that are a fixed distance or percentage distance from cluster centroids
    - Filter out outliers candidate from training dataset and assess your models performance
  - **Information Theoretic Models**: Outliers are detected as data instances that increase the complexity (minimum code length) of the dataset.
  - **High-Dimensional Outlier Detection**: Methods that search subspaces for outliers give the breakdown of distance based measures in higher dimensions (curse of dimensionality).
  - **Projection Methods**:
    - Use projection methods to summarize your data to two dimensions (such as PCA, SOM or Sammon's mapping)
    - Visualize the mapping and identify outliers by hand
    - Use proximity measures from projected values or codebook vectors to identify outliers
    - Filter out outliers candidate from training dataset and assess your models performance
  - **Methods Robust to Outliers**: An alternative strategy is to move to models that are robust to outliers. There are robust forms of regression that minimize the median least square errors rather than mean (so-called robust regression), but are more computationally intensive. There are also methods like decision trees that are robust to outliers.
- Feature Engineering A feature is an attribute that is useful or meaningful to your problem.
  - Better features means flexibility.
  - Better features means simpler models.
  - Better features means better results.
  - Iterative Process of Feature Engineering:
    - **Brainstorm features**: Really get into the problem, look at a lot of data, study feature engineering on other problems and see what you can steal.
    - **Devise features**: Depends on your problem, but you may use automatic feature extraction, manual feature construction and mixtures of the two.
    - **Select features**: Use different feature importance scorings and feature selection methods to prepare one or more "views" for your models to operate upon.
    - **Evaluate models**: Estimate model accuracy on unseen data using the chosen features.
  - Decompose Categorical Attributes
  - Decompose a Date-Time
  - Reframe Numerical Quantities
  - *KDD Cup 2010 Winner*: Feature engineering simplified the structure of the problem at the expense of creating millions of binary features. The simple structure allowed the team to use highly performant but very simple linear methods to achieve the winning predictive model.
  - 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. Examples of dimensionality reduction methods include Principal Component Analysis, Singular Value Decomposition and Sammon's Mapping.
  - Fewer attributes is desirable because it reduces the complexity of the model, and a simpler model is simpler to understand and explain. The objective of variable selection is three-fold: improving the prediction performance of the predictors, providing faster and more cost-effective predictors, and providing a better understanding of the underlying process that generated the data.
  - Feature Selection Algorithms:
    - **Filter Methods** 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:
  - **Do you have domain knowledge?** If yes, construct a better set of ad hoc"" features
  - **Are your features commensurate?** If no, consider normalizing them.
  - **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.
  - **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
  - **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.
  - **Do you need a predictor?** If no, stop
  - **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.
  - **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.
  - **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
  - **Do you want a stable solution (to improve performance and/or understanding)?** If yes, subsample your data and redo your analysis for several "bootstrap".

- Imbalanced Classes: The accuracy paradox is the name for the exact situation. It is the case where your accuracy measures tell the story that you have excellent accuracy (such as 90%), but the accuracy is only reflecting the underlying class distribution. Different ways to tackle imbalanced dataset
  - **Can You Collect More Data?** More examples of minor classes may be useful later when we look at resampling your dataset.
  - **Try Changing Your Performance Metric** Accuracy is not the metric to use when working with an imbalanced dataset. Other options are
    - Confusion Matrix: A breakdown of predictions into a table showing correct predictions (the diagonal) and the types of incorrect predictions made (what classes incorrect predictions were assigned).
    - Precision: A measure of a classifiers exactness.
    - Recall: A measure of a classifiers completeness
    - F1 Score (or F-score): A weighted average of precision and recall.
    - Kappa (or Cohen's kappa): Classification accuracy normalized by the imbalance of the classes in the data.
    - ROC Curves: Like precision and recall, accuracy is divided into sensitivity and specificity and models can be chosen based on the balance thresholds of these values.
  - **Try Resampling Your Dataset** You can change the dataset that you use to build your predictive model to have more balanced data.
    - You can add copies of instances from the under-represented class called over-sampling (or more formally sampling with replacement), or
    - You can delete instances from the over-represented class, called under-sampling.
    - Some Rules of Thumb:
      - Consider testing under-sampling when you have an a lot data (tens- or hundreds of thousands of instances or more)
      - Consider testing over-sampling when you don't have a lot of data (tens of thousands of records or less)
      - Consider testing random and non-random (e.g. stratified) sampling schemes.
      - Consider testing different resampled ratios (e.g. you don't have to target a 1:1 ratio in a binary classification problem, try other ratios)
  - **Try Generate Synthetic Samples** A simple way to generate synthetic samples is to randomly sample the attributes from instances in the minority class.
    - You could sample them empirically within your dataset or you could use a method like Naive Bayes that can sample each attribute independently when run in reverse. You will have more and different data, but the non-linear relationships between the attributes may not be preserved.
    - There are systematic algorithms that you can use to generate synthetic samples. The most popular of such algorithms is called SMOTE or the Synthetic Minority Over-sampling Technique.
  - **Try Different Algorithms** Decision trees often perform well on imbalanced datasets. If in doubt, try a few popular decision tree algorithms like C4.5, C5.0, CART, and Random Forest.
  - **Try Penalized Models** Penalized classification imposes an additional cost on the model for making classification mistakes on the minority class during training. These penalties can bias the model to pay more attention to the minority class. Often the handling of class penalties or weights are specialized to the learning algorithm. There are penalized versions of algorithms such as penalized-SVM and penalized-LDA.
  - **Try a Different Perspective** Two approaches might like to consider are **anomaly detection** and **change detection**.

        - Anomaly detection is the detection of rare events. This might be a machine malfunction indicated through its vibrations or a malicious activity by a program indicated by it's sequence of system calls. The events are rare and when compared to normal operation. This shift in thinking considers the minor class as the outliers class which might help you think of new ways to separate and classify samples.
        - Change detection is similar to anomaly detection except rather than looking for an anomaly it is looking for a change or difference. This might be a change in behavior of a user as observed by usage patterns or bank transactions.
    - **Try Getting Creative** Think about how to break it down into smaller problems that are more tractable.
- **Spot Check Algorithms**
    - I use 10 fold cross validation in my test harnesses by default. All experiments (algorithm and dataset combinations) are repeated 10 times and the mean and standard deviation of the accuracy is collected and reported. I also use statistical significance tests to flush out meaningful results from noise. Box-plots are very useful for summarizing the distribution of accuracy results for each algorithm and dataset pair.
    - I spot check algorithms, which means loading up a bunch of standard machine learning algorithms into my test harness and performing a formal experiment. I typically run 10-20 standard algorithms from all the major algorithm families across all the transformed and scaled versions of the dataset I have prepared.
    - Test Harness: The test harness is the data you will train and test an algorithm against and the performance measure you will use to assess its performance. It is important to define your test harness well so that you can focus on evaluating different algorithms and thinking deeply about the problem. If a variety of different learning algorithms universally perform poorly on the problem, it may be an indication of a lack of structure available to algorithms to learn. This may be because there actually is a lack of learnable structure in the selected data or it may be an opportunity to try different transforms to expose the structure to the learning algorithms.
    - Test and Train Datasets: An algorithm will be trained on the training dataset and will be evaluated against the test set. This may be as simple as selecting a random split of data (66% for training, 34% for testing) or may involve more complicated sampling methods.
    - Grid-searching and random searching of algorithm parameters can add new configurations en-mass or as the results of the searching process.
    - Cross Validation:
        - A more sophisticated approach than using a test and train dataset is to use the entire transformed dataset to train and test a given algorithm. A method you could use in your test harness that does this is called cross validation.
        - It first involves separating the dataset into a number of equally sized groups of instances (called folds). The model is then trained on all folds exception one that was left out and the prepared model is tested on that left out fold. The process is repeated so that each fold get's an opportunity at being left out and acting as the test dataset. Finally, the performance measures are averaged across all folds to estimate the capability of the algorithm on the problem.
        - The number of folds can vary based on the size of your dataset, but common numbers are 3, 5, 7 and 10 folds. The goal is to have a good balance between the size and representation of data in your train and test sets.
        - When you're just getting started, stick with a simple split of train and test data (such as 66%/34%) and move onto cross validation once you have more confidence.
    - Testing Algorithms:
        - The best first algorithm to spot check is a random. Plug in a random number generator to generate predictions in the appropriate range. This should be the worst "algorithm result" you achieve and will be the measure by which all improvements can be assessed.
        - Select 5-10 (sometimes 50+) standard algorithms that are appropriate for your problem and run them through your test harness. By standard algorithms, I mean popular methods no special configurations. Appropriate for your problem means that the algorithms can handle regression if you have a regression problem.
    - Tips for Spot-Checking Algorithms:
        - **Algorithm Diversity**: You want a good mix of algorithm types. I like to include instance based methods (live LVQ and knn), functions and kernels (like neural nets, regression and SVM), rule systems (like Decision Table and RIPPER) and decision trees (like CART, ID3 and C4.5).
        - **Best Foot Forward**: Each algorithm needs to be given a chance to put it's best foot forward. This does not mean performing a sensitivity analysis on the parameters of each algorithm, but using experiments and heuristics to give each algorithm a fair chance. For example if kNN is in the mix, give it 3 chances with k values of 1, 5 and 7.
        - **Formal Experiment**: Don't play. There is a huge temptation to try lots of different things in an informal manner, to play around with algorithms on your problem. The idea of spot-checking is to get to the methods that do well on the problem, fast. Design the experiment, run it, then analyze the results. Be methodical. I like to rank algorithms by their statistical significant wins (in pairwise comparisons) and take the top 3-5 as a basis for tuning.
        - **Jumping-off Point**: The best performing algorithms are a starting point not the solution to the problem. The algorithms that are shown to be effective may not be the best algorithms for the job. They are most likely to be useful pointers to types of algorithms that perform well on the problem. For example, if kNN does well, consider follow-up experiments on all the instance based methods and variations of kNN you can think of.
        - **Build Your Short-list**: As you learn and try many different algorithms you can add new algorithms to the suite of algorithms that you use in a spot-check experiment. When I discover a particularly powerful configuration of an algorithm, I like to generalize it and include it in my suite, making my suite more robust for the next problem.
    - The top 10 algorithms for data mining:
        - C4.5 This is a decision tree algorithm and includes descendent methods like the famous C5.0 and ID3 algorithms.
        - k-means. The go-to clustering algorithm.
        - Support Vector Machines. This is really a huge field of study.
        - Apriori. This is the go-to algorithm for rule extraction.
        - EM. Along with k-means, go-to clustering algorithm.
        - PageRank. I rarely touch graph-based problems.
        - AdaBoost. This is really the family of boosting ensemble methods.
        - knn (k-nearest neighbor). Simple and effective instance-based method.
        - Naive Bayes. Simple and robust use of Bayes theorem on data.
        - CART (classification and regression trees) another tree-based method.
    - How to test a given algorithm:
        - Split Test: Split data into 66 and 34 % for training and testing.
        - Multiple Split Tests: Run split test multiple times (around 10) and take an average and standard deviation of the result.
        - Cross Validation: use K-fold cross validation (k=10). checks how robust an algorithm is on a given dataset.

- Multiple Cross Validation: run cross validation test multiple times and take an average and standard deviation of the result. A solution to comparing algorithm performance measures when using multiple runs of k-fold cross validation is to use statistical significance tests (like the Student's t-test). The results from multiple runs of k-fold cross validation is a list of numbers. We like to summarize these numbers using the mean and standard deviation. You can think of these numbers as a sample from an underlying population. A statistical significance test answers the question: are two samples drawn from the same population? (no difference). If the answer is "yes", then, even if the mean and standard deviations differ, the difference can be said to be not statistically significant. We can use statistical significance tests to give meaning to the differences (or lack there of) between algorithm results when using multiple runs (like multiple runs of k-fold cross validation with different random number seeds). This can when we want to make accurate claims about results (algorithm A was better than algorithm B and the difference was statistically significant). When in doubt, use k-fold cross validation (k=10) and use multiple runs of k-fold cross validation with statistical significance tests when you want to meaningfully compare algorithms on your dataset.

- **Improve Results**
  - After spot checking, it's time to squeeze out the best result from the rig. I do this by running an automated sensitivity analysis on the parameters of the top performing algorithms. I also design and run experiments using standard ensemble methods of the top performing algorithms.
  - Statistical significance of results is critical here.
  - **Algorithm Tuning** where discovering the best models is treated like a search problem through model parameter space.
  - **Ensemble Methods** where the predictions made by multiple models are combined. It is a good idea to get into ensemble methods after you have exhausted more traditional methods.
    - **Bagging**: Known more formally as Bootstrapped Aggregation is where the same algorithm has different perspectives on the problem by being trained on different subsets of the training data.
    - **Boosting**: Different algorithms are trained on the same training data.
    - **Blending**: Known more formally as Stacked Generalization or Stacking is where a variety of models whose predictions are taken as input to a new model that learns how to combine the predictions into an overall prediction.
  - **Extreme Feature Engineering** where the attribute decomposition and aggregation seen in data preparation is pushed to the limits. Take attributes and decompose them widely into multiple features. Technically, what you are doing with this strategy is reducing dependencies and non-linear relationships into simpler independent linear relationships.
    - **Categorical**: You have a categorical attribute that had the values [red, green blue], you could split that into 3 binary attributes of red, green and blue and give each instance a 1 or 0 value for each.
    - **Real**: You have a real valued quantity that has values ranging from 0 to 1000. You could create 10 binary attributes, each representing a bin of values (0-99 for bin 1, 100-199 for bin 2, etc.) and assign each instance a binary value (1/0) for the bins.
  - Machine Learning Performance Improvement The gains often get smaller the further you go down the list.

    - Improve Performance With Data: Create new and different perspectives on your data in order to best expose the structure of the underlying problem to the learning algorithms (basically, creating and exposing new feature spaces).

      - **Get More Data.**
      - **Invent More Data.** This is also related to adding noise, what we used to call adding jitter. It can act like a regularization method to curb overfitting the training dataset.
        - If your data are vectors of numbers, create randomly modified versions of existing vectors.
        - If your data are images, create randomly modified versions of existing images.
          - For example, with photograph image data, you can get big gains by randomly shifting and rotating existing images. It improves the generalization of the model to such transforms in the data if they are to be expected in new data.
        - If your data are text, create randomly modified versions of existing text.
      - **Clean Your Data**.
      - **Resample Data.** *Can you resample data to change the size or distribution?* Perhaps you can use a much smaller sample of data for your experiments to speed things up or over-sample or under-sample observations of a specific type to better represent them in your dataset.
      - **Reframe Your Problem.** *Can you change the type of prediction problem you are solving?* Reframe your data as a regression, binary or multiclass classification, time series, anomaly detection, rating, recommender, etc. type problem.
      - **Rescale Your Data.** *Can you rescale numeric input variables?* Normalization and standardization of input data can result in a lift in performance on algorithms that use weighted inputs or distance measures.
        - Normalized to 0 to 1
          - Normalization is a good technique to use when you do not know the distribution of your data or when you know the distribution is not Gaussian (a bell curve).
          - Normalization is useful when your data has varying scales and the algorithm you are using does not make assumptions about the distribution of your data, such as k-nearest neighbors and artificial neural networks.
        - Rescaled to -1 to 1
          - You can use other scales such as -1 to 1, which is useful when using support vector machines and adaboost.
        - Standardized
          - Data standardization is the process of rescaling one or more attributes so that they have a mean value of 0 and a standard deviation of 1.
          - Standardization assumes that your data has a Gaussian (bell curve) distribution. This does not strictly have to be true, but the technique is more effective if your attribute distribution is Gaussian.
          - Standardization is useful when your data has varying scales and the algorithm you are using does make assumptions about your data having a Gaussian distribution, such as linear regression, logistic regression and linear discriminant analysis.
        - There are other methods for keeping numbers small in your network such as normalizing activation and weights.
      - **Transform Your Data**. *Can you reshape your data distribution?* Making input data more Gaussian or passing it through an exponential function may better expose features in the data to a learning algorithm.
        - Guesstimate the univariate distribution of each column (visualise data).

          - Does a column look like a skewed Gaussian, consider adjusting the skew with a Box-Cox transform.
          - Does a column look like an exponential distribution, consider a log transform.

        - Does a column look like it has some features, but they are being clobbered by something obvious, try squaring, or square-rooting.
        - Can you make a feature discrete or binned in some way to better emphasize some feature.
        - Can you pre-process data with a projection method like PCA?
        - Can you aggregate multiple attributes into a single value?
        - Can you expose some interesting aspect of the problem with a new boolean flag?
        - Can you explore temporal or other structure in some other way?
    - **Project Your Data**: *Can you project your data into a lower dimensional space?* You can use an unsupervised clustering or projection method to create an entirely new compressed representation of your dataset.
    - **Feature Selection**. *Are all input variables equally important?* Use feature selection and feature importance methods to create new views of your data to explore with modeling algorithms.
    - **Feature Engineering**. *Can you create and add new data features?* Perhaps there are attributes that can be decomposed into multiple new values (like categories, dates or strings) or attributes that can be aggregated to signify an event (like a count, binary flag or statistical summary).
- Improve Performance With Algorithms Identify the algorithms and data representations that perform above a baseline of performance and better than average.

    - **Resampling Method**. *What resampling method is used to estimate skill on new data?* Use a method and configuration that makes the best use of available data. The k-fold cross-validation method with a hold out validation dataset might be a best practice. The train-test split and k-fold cross validation are called resampling methods. Resampling methods are statistical procedures for sampling a dataset and estimating an unknown quantity. While finalising a model we train it using all the data we have, using the parameters we have finalised during training and testing, during convergence of accuracy.
    - **Evaluation Metric**. *What metric is used to evaluate the skill of predictions?* Use a metric that best captures the requirements of the problem and the domain. It probably isn't classification accuracy.
    - **Baseline Performance**. *What is the baseline performance for comparing algorithms?* Use a random algorithm or a zero rule algorithm (predict mean or mode) to establish a baseline by which to rank all evaluated algorithms.
    - **Spot Check Linear Algorithms**. *What linear algorithms work well?* Linear methods are often more biased, are easy to understand and are fast to train. They are preferred if you can achieve good results. Evaluate a diverse suite of linear methods.
    - **Spot Check Nonlinear Algorithms**. *What nonlinear algorithms work well?* Nonlinear algorithms often require more data, have greater complexity but can achieve better performance. Evaluate a diverse suite of nonlinear methods.
    - **Steal from Literature**.
    - **Standard Configurations**. *What are the standard configurations for the algorithms being evaluated?* Each algorithm needs an opportunity to do well on your problem. This does not mean tune the parameters (yet) but it does mean to investigate how to configure each algorithm well and give it a fighting chance in the algorithm bake-off.
    - Spot-check a suite of top methods and see which fair well and which do not.

        - Evaluate some linear methods like logistic regression and linear discriminate analysis.
        - Evaluate some tree methods like CART, Random Forest and Gradient Boosting.
        - Evaluate some instance methods like SVM and kNN.
        - Evaluate some other neural network methods like LVQ, MLP, CNN, LSTM, hybrids, etc.
- Improve Performance With Algorithm Tuning

    - **Diagnostics**. *What diagnostics and you review about your algorithm?* Perhaps you can review learning curves to understand whether the method is over or underfitting the problem, and then correct. Different algorithms may offer different visualizations and diagnostics. Review what the algorithm is predicting right and wrong.
    - **Try Intuition**.
    - **Steal from Literature**.
    - **Random Search**. *What parameters can use random search?* Perhaps you can use random search of algorithm hyperparameters to expose configurations that you would never think to try.
    - **Grid Search**. *What parameters can use grid search?* Perhaps there are grids of standard hyperparameter values that you can enumerate to find good configurations, then repeat the process with finer and finer grids.
    - **Optimize**. *What parameters can you optimize?* Perhaps there are parameters like structure or learning rate that can be tuned using a direct search procedure (like pattern search) or stochastic optimization (like a genetic algorithm).
    - **Alternate Implementations**. *What other implementations of the algorithm are available?* Perhaps an alternate implementation of the method can achieve better results on the same data. Each algorithm has a myriad of micro-decisions that must be made by the algorithm implementor. Some of these decisions may affect skill on your problem.
    - **Algorithm Extensions**. *What are common extensions to the algorithm?* Perhaps you can lift performance by evaluating common or standard extensions to the method. This may require implementation work.
    - **Algorithm Customizations**. *What customizations can be made to the algorithm for your specific case?* Perhaps there are modifications that you can make to the algorithm for your data, from loss function, internal optimization methods to algorithm specific decisions.
    - **Contact Experts**.
    - Some ideas on tuning your neural network algorithms
        - Diagnostics Is your model overfitting or underfitting?

            - A quick way to get insight into the learning behavior of your model is to evaluate it on the training and a validation dataset each epoch, and plot the results.
                - If training is much better than the validation set, you are probably overfitting and you can use techniques like regularization.
                - If training and validation are both low, you are probably underfitting and you can probably increase the capacity of your network and train more or longer.
                - If there is an inflection point when training goes above the validation, you might be able to use early stopping.
        - Weight Initialization Changing the weight initialization method is closely tied with the activation function and even the optimization function.

            - Try all the different initialization methods offered and see if one is better with all else held constant.
            - Try pre-learning with an unsupervised method like an autoencoder.

        - Try taking an existing model and retraining a new input and output layer for your problem (transfer learning).
    - Learning Rate Learning rate is coupled with the number of training epochs, batch size and optimization method.
        - Experiment with very large and very small learning rates.
        - Grid search common learning rate values from the literature and see how far you can push the network.
        - Try a learning rate that decreases over epochs.
        - Try a learning rate that drops every fixed number of epochs by a percentage.
        - Try adding a momentum term then grid search learning rate and momentum together.
    - Activation Functions Possible options are rectifier activation functions, sigmoid and tanh, then a softmax, linear or sigmoid on the output layer.
    - Network Topology How many layers and how many neurons do you need?
        - Try one hidden layer with a lot of neurons (wide).
        - Try a deep network with few neurons per layer (deep).
        - Try combinations of the above.
        - Try architectures from recent papers on problems similar to yours.
        - Try topology patterns (fan out then in) and rules of thumb from books and papers
    - Batches and Epochs The batch size defines the gradient and how often to update weights. An epoch is the entire training data exposed to the network, batch-by-batch.
        - Try batch size equal to training data size, memory depending (batch learning).
        - Try a batch size of one (online learning).
        - Try a grid search of different mini-batch sizes (8, 16, 32, …).
        - Try training for a few epochs and for a heck of a lot of epochs.
    - Regularization Regularization is a great approach to curb overfitting the training data. Dropout randomly skips neurons during training, forcing others in the layer to pick up the slack.
        - Grid search different dropout percentages.
        - Experiment with dropout in the input, hidden and output layers.
    - Optimization and Loss Stochastic Gradient Descent is the default. Get the most out of it first, with different learning rates, momentum and learning rate schedules. You can also explore other optimization algorithms such as the more traditional (Levenberg-Marquardt) and the less so (genetic algorithms).
        - ADAM
        - RMSprop
    - Early Stopping Early stopping is a type of regularization to curb overfitting of the training data and requires that you monitor the performance of the model on training and a held validation datasets, each epoch. Once performance on the validation dataset starts to degrade, training can stop.
- Improve Performance With Ensembles You can often get good performance from combining the predictions from multiple "good enough" models rather than from multiple highly tuned (and fragile) models.
    - **Blend Model Predictions**. *Can you combine the predictions from multiple models directly?* Perhaps you could use the same or different algorithms to make multiple models. Take the mean or mode from the predictions of multiple well-performing models.
    - **Blend Data Representations**. *Can you combine predictions from models trained on different data representations?* Yo u may have many different projections of your problem which can be used to train well-performing algorithms, whose predictions can then be combined.
    - **Blend Data Samples**. *Can you combine models trained on different views of your data?* Perhaps you can create multiple subsamples of your training data and train a well-performing algorithm, then combine predictions. This is called bootstrap aggregation or bagging and works best when the predictions from each model are skillful but in different ways (uncorrelated).
    - **Correct Predictions**. *Can you correct the predictions of well-performing models?* Perhaps you can explicitly correct predictions or use a method like boosting to learn how to correct prediction errors.
    - **Learn to Combine or Stacking or Stacked Generalization**. *Can you use a new model to learn how to best combine the predictions from multiple well-performing models?* This is called stacked generalization or stacking and often works well when the submodels are skillful but in different ways and the aggregator model is a simple linear weighting of the predictions. This process can be repeated multiple layers deep.

- **Present Results**
    - Report Results
        - **Context (Why)** Define the environment in which the problem exists and set up the motivation for the research question.
        - **Problem (Question)** Concisely describe the problem as a question that you went out and answered.
        - **Solution (Answer)** Concisely describe the solution as an answer to the question you posed in the previous section. Be specific.
        - **Findings** Bulleted lists of discoveries you made along the way that interests the audience. They may be discoveries in the data, methods that did or did not work or the model performance benefits you achieved along your journey.
        - **Limitations** Consider where the model does not work or questions that the model does not answer. Do not shy away from these questions, defining where the model excels is more trusted if you can define where it does not excel.
        - **Conclusions (Why+Question+Answer)** Revisit the "why", research question and the answer you discovered in a tight little package that is easy to remember and repeat for yourself and others.
    - Operationalize
        - **Algorithm Implementation** Think very hard about the dependencies and technical debt you may be creating by putting such an implementation directly into production. Consider locating a production-level library that supports the method you wish to use. You may have to repeat the process of algorithm tuning if you switch to a production level library at this point.
        - **Model Tests** Write automated tests that verify that the model can be constructed and achieve a minimum level of performance repeatedly. Also write tests for any data preparation steps. You may wish to control the randomness used by the algorithm (random number seeds) for each unit test run so that tests are 100% reproducible.

- **Tracking** Add infrastructure to monitor the performance of the model over time and raise alarms if accuracy drops below a minimum level. Tracking may occur in real-time or with samples of live data on a re-created model in a separate environment. A raised alarm may be an indication that that structure learned by the model in the data have changed (concept drift) and that the model may need to be updated or tuned. There are some model types that can perform online learning and update themselves. Think carefully in allowing models to update themselves in a production environment.

---

5 Model Deployment Best Practices

- **Specify Performance Requirements** You need to clearly spell out what constitutes good and bad performance. This maybe as accuracy or false positives or whatever metrics are important to the business. Use the current model you have developed as the baseline numbers. Do not proceed until you have agreed upon minimum, mean or a performance range expectation.
- **Separate Prediction Algorithm From Model Coefficients** You can choose to deploy your model using that library or re-implement the predictive aspect of the model in your software. It is good practice to separate the algorithm that makes predictions from the model internals. That is the specific coefficients or structure within the model learned from your training data.
    - Select or Implement The Prediction Algorithm: The software used to make predictions is just like all the other software in your application. Implement it well, write unit tests, make it robust.
    - Serialize Your Model Coefficients: Store it in an external file with the software project. Version it. Treat configuration like code because it can just as easily break your project.
- **Develop Automated Tests For Your Model** You need automated tests to prove that your model works as you expect. I strongly recommend contriving test cases that you understand well, in addition to any raw datasets from the domain you want to include. I also strongly recommend gathering outlier and interesting cases from operations over time that produce unexpected results (or break the system). These should be understood and added to the regression test suite. Run the regression tests after each code change and before each release. Run them nightly.
    - Collect or contribute a small sample of data on which to make predictions.
    - Use the production algorithm code and configuration to make predictions.
    - Confirm the results are expected in the test.
- **Develop Back-Testing and Now-Testing Infrastructure**
    - Back-Testing: You want to automate the evaluation of the production model with a specified configuration on a large corpus of historical data.
    - Now-Testing: Perhaps it's the data from today, this week or this month. The idea is to get an early warning that the production model may be faltering.
- **Challenge Then Trial Model Updates** Maybe you devise a whole new algorithm which requires new code and new config. Revisit all of the above points. Evaluate the performance of the new model using the Back-Test and Now-Test infrastructure.

- ## System Design
    - **Raw Data**: The raw data source (database access) or files
    - **Feature Space selection**: Views on the problem defined as queries or flat files
    - **Data Partition**: Splitting of data views into cross-validation folds, test/train folds and any other folds needed to evaluate models and make predictions for the competition.
    - **Analysis Reports**: Summarization of a data view using descriptive statistics and plots.
    - **Models**: Machine learning algorithm and configuration that together with input data is used to construct a model just-in-time.
    - **Model Outputs**: The raw results for models on all data partitions for all data views.
    - **Blends**: Ensemble algorithms and configurations designed to create blends of model outputs for all data partitions on all data views.
    - **Scoreboard**: Local scoreboard that describes all completed runs and their scores that can be sorted and summarized.
    - **Predictions**: Submittable predictions for uploading to competition leader boards, etc.

Algorithms

- **zeroR algorithm** as baseline algorithm for comparison: The ZeroR algorithm also called the Zero Rule is an algorithm that you can use to calculate a baseline of performance for all algorithms on your dataset. It is the “worst” result and any algorithm that shows a better performance has some skill on your problem. The ZeroR algorithm selects the majority class in the dataset and uses that to make all predictions.
    - On a classification algorithm, the ZeroR algorithm will always predict the most abundant category. If the dataset has an equal number of classes, it will predict the first category value.
    - For regression problems, the ZeroR algorithm will always predict the mean output value.
- Top five **classification algorithms** (as defined in Weka software)
    - Logistic Regression
    - Naive Bayes
    - k-Nearest Neighbors
    - Classification and Regression Trees
    - Support Vector Machines
- Top five **regression algorithms** (as defined in Weka software)
    - Linear Regression
    - Support Vector Regression
    - k-Nearest Neighbors
    - Classification and Regression Trees
    - Artificial Neural Network
- Top five **ensemble algorithms** (as defined in Weka software)
    - Bagging
    - Random Forest
    - AdaBoost
    - Voting
    - Stacking
- Machine Learning
    - Reinforcement Learning
    - Supervised Learning
        - Regression
            - Decision Tree
            - Linear Regression
            - Logistic Regression
        - Classification
            - Navie Bayes
            - SVM
            - K-Nearest Neighbour
    - Unsupervised Learning
        - Clustering
            - K-Means
            - Mean Shift

        - K-Medoids
    - Dimensionality Reduction
        - PCA
        - Feature Selection
        - Linear Discriminant Analysis
- Parametric vs Non Parametric Algorithms
    - Parametric:
        - Algorithms that simplify the function to a known form are called parametric machine learning algorithms.
        - A learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples) is called a parametric model. No matter how much data you throw at a parametric model, it won't change its mind about how many parameters it needs.
        - Parametric methods make large assumptions about the mapping of the input variables to the output variable and in turn are faster to train, require less data but may not be as powerful.
        - Some more examples of parametric machine learning algorithms include:
            - Logistic Regression
            - Linear Discriminant Analysis
            - Perceptron
            - Naive Bayes
            - Simple Neural Networks
        - Benefits of Parametric Machine Learning Algorithms:
            - **Simpler**: These methods are easier to understand and interpret results.
            - **Speed**: Parametric models are very fast to learn from data.
            - **Less Data**: They do not require as much training data and can work well even if the fit to the data is not perfect.
        - Limitations of Parametric Machine Learning Algorithms:
            - **Constrained**: By choosing a functional form these methods are highly constrained to the specified form.
            - **Limited Complexity**: The methods are more suited to simpler problems.
            - **Poor Fit**: In practice the methods are unlikely to match the underlying mapping function.
    - Non Parametric:
        - Algorithms that do not make strong assumptions about the form of the mapping function are called nonparametric machine learning algorithms. By not making assumptions, they are free to learn any functional form from the training data.
        - Nonparametric methods are good when you have a lot of data and no prior knowledge, and when you don't want to worry too much about choosing just the right features.
        - Nonparametric methods make few or no assumptions about the target function and in turn require a lot more data, are slower to train and have a higher model complexity but can result in more powerful models.
        - Some more examples of popular nonparametric machine learning algorithms are:
            - k-Nearest Neighbors
            - Decision Trees like CART and C4.5
            - Support Vector Machines
        - Benefits of Nonparametric Machine Learning Algorithms:
            - **Flexibility**: Capable of fitting a large number of functional forms.
            - **Power**: No assumptions (or weak assumptions) about the underlying function.
            - **Performance**: Can result in higher performance models for prediction.
        - Limitations of Nonparametric Machine Learning Algorithms:
            - **More data**: Require a lot more training data to estimate the mapping function.
            - **Slower**: A lot slower to train as they often have far more parameters to train.
            - **Overfitting**: More of a risk to overfit the training data and it is harder to explain why specific predictions are made.
- Common classes of problems in ML
    - **Classification**: Data is labelled meaning it is assigned a class, for example spam/non-spam or fraud/non-fraud. The decision being modelled is to assign labels to new unlabelled pieces of data. This can be thought of as a discrimination problem, modelling the differences or similarities between groups.
    - **Regression**: Data is labelled with a real value (think floating point) rather then a label. Examples that are easy to understand are time series data like the price of a stock over time, The decision being modelled is what value to predict for new unpredicted data.
    - **Clustering**: Data is not labelled, but can be divided into groups based on similarity and other measures of natural structure in the data. An example from the above list would be organising pictures by faces without names, where the human user has to assign names to groups, like iPhoto on the Mac.
    - **Rule Extraction**: Data is used as the basis for the extraction of propositional rules (antecedent/consequent aka *if-then*). Such rules may, but are typically not directed, meaning that the methods discover statistically supportable relationships between attributes in the data, not necessarily involving something that is being predicted. An example is the discovery of the relationship between the purchase of beer and diapers (this is data mining folk-law, true or not, it's illustrative of the desire and opportunity).
- Two categories of algorithms
    - The first is a grouping of algorithms by their **learning style**.
        - **Supervised Learning**
            - Example problems are classification and regression.
            - Example algorithms include: Logistic Regression and the Back Propagation Neural Network.
        - **Unsupervised Learning**
            - A model is prepared by deducing structures present in the input data. This may be to extract general rules. It may be through a mathematical process to systematically reduce redundancy, or it may be to organize data by similarity.
            - Example problems are clustering, dimensionality reduction and association rule learning.
            - Example algorithms include: the Apriori algorithm and K-Means.
        - **Semi-Supervised Learning**
            - There is a desired prediction problem but the model must learn the structures to organize the data as well as make predictions.
            - Example problems are classification and regression.

- The second is a grouping of algorithms by their **similarity** in form or function (like grouping similar animals together).
    - Algorithms are often grouped by similarity in terms of their function (how they work). For example, tree-based methods, and neural network inspired methods.

- Types of ML Algorithms
    - Regression Algorithms Regression is concerned with modeling the relationship between variables that is iteratively refined using a measure of error in the predictions made by the model.
        - Ordinary Least Squares Regression (OLSR)
        - Linear Regression
        - Logistic Regression
        - Stepwise Regression
        - Multivariate Adaptive Regression Splines (MARS)
        - Locally Estimated Scatterplot Smoothing (LOESS)
    - Instance-based Algorithms (aka winner-take-all methods, memory-based learning) Instance-based learning model is a decision problem with instances or examples of training data that are deemed important or required to the model.
        - k-Nearest Neighbor (kNN)
        - Learning Vector Quantization (LVQ)
        - Self-Organizing Map (SOM)
        - Locally Weighted Learning (LWL)
        - Support Vector Machines (SVM)
    - Regularization Algorithms An extension made to another method (typically regression methods) that penalizes models based on their complexity, favoring simpler models that are also better at generalizing.
        - Ridge Regression
        - Least Absolute Shrinkage and Selection Operator (LASSO)
        - Elastic Net
        - Least-Angle Regression (LARS)
    - Decision Tree Algorithms Decision tree methods construct a model of decisions made based on actual values of attributes in the data. Decisions fork in tree structures until a prediction decision is made for a given record. Decision trees are trained on data for classification and regression problems. - Learning tradeoff between Bias and Variance - http://www.r2d3.us/visual-intro-to-machine-learning-part-2
        - Classification and Regression Tree (CART)
        - Iterative Dichotomiser 3 (ID3)
        - C4.5 and C5.0 (different versions of a powerful approach)
        - Chi-squared Automatic Interaction Detection (CHAID)
        - Decision Stump
        - M5
        - Conditional Decision Trees
    - Bayesian Algorithms Bayesian methods are those that explicitly apply Bayes' Theorem for problems such as classification and regression.
        - Naive Bayes
        - Gaussian Naive Bayes
        - Multinomial Naive Bayes
        - Averaged One-Dependence Estimators (AODE)
        - Bayesian Belief Network (BBN)
        - Bayesian Network (BN)
    - Clustering Algorithms Clustering, like regression, describes the class of problem and the class of methods. Clustering methods are typically organized by the modeling approaches such as centroid-based and hierarchal. All methods are concerned with using the inherent structures in the data to best organize the data into groups of maximum commonality.
        - k-Means
        - k-Medians
        - Expectation Maximisation (EM)
        - Hierarchical Clustering
    - Association Rule Learning Algorithms Association rule learning methods extract rules that best explain observed relationships between variables in data.
        - Apriori algorithm
        - Eclat algorithm
    - Artificial Neural Network Algorithms Artificial Neural Networks are models that are inspired by the structure and/or function of biological neural networks. They are a class of pattern matching that are commonly used for regression and classification problems but are really an enormous subfield comprised of hundreds of algorithms and variations for all manner of problem types.
        - Perceptron
        - Multilayer Perceptrons (MLP)
        - Back-Propagation
        - Stochastic Gradient Descent
        - Hopfield Network
        - Radial Basis Function Network (RBFN)
    - Deep Learning Algorithms Deep Learning methods are a modern update to Artificial Neural Networks that exploit abundant cheap computation. They are concerned with building much larger and more complex neural networks and, as commented on above, many methods are concerned with very large datasets of labelled analog data, such as image, text. audio, and video.
        - Convolutional Neural Network (CNN)
        - Recurrent Neural Networks (RNNs)
        - Long Short-Term Memory Networks (LSTMs)
        - Stacked Auto-Encoders
        - Deep Boltzmann Machine (DBM)
        - Deep Belief Networks (DBN)

- Dimensionality Reduction Algorithms Like clustering methods, dimensionality reduction seek and exploit the inherent structure in the data, but in this case in an unsupervised manner or order to summarize or describe data using less information. This can be useful to visualize dimensional data or to simplify data which can then be used in a supervised learning method. Many of these methods can be adapted for use in classification and regression.
    - Principal Component Analysis (PCA)
    - Principal Component Regression (PCR)
    - Partial Least Squares Regression (PLSR)
    - Sammon Mapping
    - Multidimensional Scaling (MDS)
    - Projection Pursuit
    - Linear Discriminant Analysis (LDA)
    - Mixture Discriminant Analysis (MDA)
    - Quadratic Discriminant Analysis (QDA)
    - Flexible Discriminant Analysis (FDA)
- Ensemble Algorithms Ensemble methods are models composed of multiple weaker models that are independently trained and whose predictions are combined in some way to make the overall prediction. Much effort is put into what types of weak learners to combine and the ways in which to combine them. This is a very powerful class of techniques and as such is very popular.
    - Boosting
    - Bootstrapped Aggregation (Bagging)
    - AdaBoost
    - Weighted Average (Blending)
    - Stacked Generalization (Stacking)
    - Gradient Boosting Machines (GBM)
    - Gradient Boosted Regression Trees (GBRT)
    - Random Forest

Tutorials

Topic	Source
TensorFlow 2.0	
Machine Learning Mastery	https://machinelearningmastery.com/