Business Intelligence and Data Analytics

Precursor:

In this document, we explore certain algorithms and methods used to analyze large sets of data and create prediction models that can extrapolate or interpolate new data. I will cover two forms of learning – supervised learning, and unsupervised learning. Note that this document is not interested in neural networks, which are what most image recognition and generative text artificial intelligences use. Neural networks and the type of learning that we will be covering in this document both fall under the umbrella of Machine Learning (ML).

Intro:

There are many times in Computer Science when we need to analyse large sets of data. You can imagine that the Government would be interested in trends amongst the general population, or perhaps a corporation that would like to visualize and predict future sales trends. Machine learning is by no means a new technology, but it has improved drastically in the last couple of years thanks to Moore’s law and the advancements in both computing power and cost effectiveness of digital storage. In terms of mathematical complexity, ML is actually quite simple, it just requires a huge amount of computations due to the (typically) recursive nature of ML algorithms. We will cover some of these algorithms in this document. Within the course that I will be referencing my material from, we used the University of Waikato’s machine learning tool, Weka. Weka is a free tool written in Java for analysing large datasets. It provides all of the tools and plugins required for most, if not all, the material I’ll be covering. That being said, there are many libraries out there for languages such as MatLab, R, and Python, which are just as suitable to handle these same concepts. Therefore, I will try to focus more on theory rather than the practical application, which I leave up to you, the reader.

CRISP-DM:

The Cross-Industry Standard Process for Data Mining (rolls right off the tongue), is a standard that is, well, cross-industry for data mining. It outlines the high-level recommended approach for data analysis. CRISP-DM is split into 6 primary phases, each of which have sub-phases. Note that not all steps in the process may be applicable to your specific scenario. The primary 6 phases are as follows:

* **Business Understanding:** This phase occurs prior to data analysis. We first want to understand what the business objective is i.e., what our goal is from a business perspective. We want to assess the situation and produce a project plan.
* **Data Understanding:** Also performed before data analysis, we first make sure that we understand what it is we’re actually analysing. We might be given a dataset that we know nothing about, so it is important that we know what sorts of data exist in the set and why it is required.
* **Data Preparation:** In data preparation, we do things such as remove data that does not pertain to our business objective. We can also (in some cases) construct new data, or join multiple related datasets together.
* **Modeling:** In modeling we determine which algorithms are most suitable for operations upon our dataset. We build the actual learning model, which is essentially a program with predictive power.
* **Evaluation:** Once the predictive model is built, we assess its accuracy and reliability. This can be done in numerous ways, which we will touch upon later.
* **Deployment:** This phase is a sort of post-analysis phase, where we deploy the model to be used for whichever purpose it was created to serve. We monitor and maintain it, and if applicable, write a report or create a project review.

Now that I’ve provided a general overview of each of the 6 primary phases in CRISP-DM, here is a condensed version of each phase and the sub-phases for each:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Business Understanding | Data Understanding | Data Preparation | Modeling | Evaluation | Deployment |
| Determine Business Objectives | Collect Initial Data | Select Data | Select Modeling Technique | Evaluate Results | Plan Deployment |
| Assess Situation | Describe Data | Clean Data | Generate Test Design | Review Process | Plan Monitoring & Maintenance |
| Determine Data Mining Goals | Explore Data | Construct Data | Build Model | Determine Next Steps | Product Final Report |
| Produce Project Plan | Verify Data Quality | Integrate Data | Assess Model |  | Review Project |
|  |  | Format Data |  |  |  |

ARFF:

Although I said that I would try to avoid refering to Weka I will still bring up the ARFF file format. The Attribute-Relation File Format (ARFF) was also created by the Unversity of Waikato to be used with Weka. It is, in fact, identical to CSV file format (used by MS Excel), with the addition of a file header that precedes the comma separated data entries. The header must at least contain 3 tags: @RELATION, @ATTRIBUTE, and @DATA. The @RELATION tag is proceeded by the name of the relation i.e., of the dataset. The @ATTRIBUTE tag defines a column a.k.a. an attribute, and its corresponding datatype. Finally, the @DATA tag just tells the program that anything proceeding it should be interpreted as the actual raw data.

What I most prominently wanted to touch upon was the @ATTRIBUTE tag. A “column” in ARFF is referred to as an attribute. This is very much akin to an attribute in a database – it is a description of one feature of the relation/dataset (note that attributes are sometimes also called ‘features’). For instance, color, or length, or amount of ‘x’, are all attributes. Attributes are split into data types. There are essentially two broad categories of data types: Categorical types, or Numerical types. Numerical data types are either discrete or continuous. A discrete number is one which cannot be measured, but can be counted, whereas a continous number is one which cannot be counted, but can be measured. For categorical types, we also have two broad categories: Nominal and Ordinal. A nominal category is one which classifies data using names, whereas an ordinal category is one which classifies data in some order or rank. Categorical types can also be numbers, so don’t think that they *must* be strings; it is just the case that they often are strings.

Math Review!:

Now onto my personal favorite part of this course: math! We will begin with mean, median, and mode. These should be straightforward concepts for most people. That being said, perhaps your memory is as poor as mine and you require a refresher. Mean is simply taking the average out of a set of data. Given a set of data ‘X’, the formula for mean would be as follows: The ‘u’ here should really be the Greek letter mu, but I digress. If you are not comfortable with the sigma/sum operator, this may appear daunting, but really, we just take the sum of each item in set X and then divide the whole thing by the length of X. As for median and mode, there is really no mathematical formula to determine them, but median is typically the middle number for a set with an odd number of elements or the middle two numbers for a set with an even number of elements and mode is the number which appears most frequently within a set.

We looked at the formula for mean, but what if you wanted to compute the mean for a subset of your data. For instance, what if you wanted to find the mean for only the data recorded within the last 5 days (including the current day), but you wanted to do this once per day. Assuming we have yesterday’s mean for the last 5 days, rather than recomputing the mean for the last five days starting today, we just add the new data for today, subtract the expired data (from 5 days ago) and then divide by the new number of items (N). Here is a visual of the last 5 days beginning yesterday (we record two entries of data per day):

[1, 2], [3, 4], [4, 6], [5, 7], [9, 11]

And we’ve computed the average like so:

(1+2+3+4+4+6+5+7+9+11)/10 = 5.2

Now today we record two new entries:

[10, 12]

Rather than recompute the average, we add this new data to yesterday’s average, subtract the stale data from 5 days ago ([1, 2]) and divide by whatever the new value of N is (in this case N stays the same):

5.2 + [(10 + 12) – (1 + 2)] / 10

= 7.1

Next we’ll look at weighted averages. A weighted average is essentially the same formula for mean, except for each item in the set, X, we multiply Xi by some weight and instead of dividing the dividend by the number of items in the set (N), we instead divide by the sum of each weight. This is useful for when certain data in the dataset hold more or less significance towards the final outcome, and therefore, are given a larger or smaller weight respectively to try and improve the accuracy of the mean. For instance, imagine that a professor is grading a student upon multiple facets: test marks, project work, attendance, and class behavior. Now assume that the professor considered test marks to be the most important measure of success, project work second, followed by attendance, then class behavior. They may assign a weight of 0.8 to test marks, 0.5 to project work, 0.3 towards attendance, and 0.2 towards class behavior. Let’s pretend that a student receives an average of 84% on all their tests, an average of 89% on their project work, and average of 73% on attendance and 95% on behavior. A normal mean would look like the following:

(84 + 89 + 73 + 95) / 4 = 85.25

Taking the weights into consideration, it would look like this:

((84 \* 0.8) + (89 \* 0.5) + (73 \* 0.3) + (95 \* 0.2)) / (0.8 + 0.5 + 0.3 + 0.2) = 84.78

Next, we’ll review the differences between sample and population. If you’ve read my numerical computing notes, you may recall that population is the entirety of a group i.e., all values in a set, whereas a sample is a subset of the population. This is important when calculating standard deviation, which if you recall, is a measure of the spread of data i.e., how much *variance* it has (how drastically the values alternate from the mean, in other words). Let us assume a set X = [21, 37, 13, 25, 32, 8]. To calculate standard deviation, we first take the average, which ends up as 22.6667. Next, we take the average (represented as mu) and put it into the formula for standard deviation. Recall that there are two variations of the formula for SD depending on whether or not you are taking it for the entire population, or for a sample of the population. Both formulas are listed below:

**SD for Population:**

**SD for Sample:**

**Variance for Population:**

**Variance for Sample:**

Next, we review normal/Guassian distribution a.k.a. bell curves. You should hopefully know that the term “normalization” refers to dividing values in a manner that results in each value existing between a normalized range e.g. 0..1 or -1..1. To do this, we normalize the data by dividing all values by the standard distribution of the set. This has the effect of making the mean 0. The center of a bell curve must by definition be the mean of the population or sample considering that the average value is also the most probable. The x-axis of the bell curve is called the z-score. Z-score is a table of predetermined probabilities depending upon the value of x. The z-score can be any number between -3.4998 and 3.4998. As the z-score approaches either of these limits, the probability of the data falling under the bell curve increases logarithmically until the function’s limit is reached. It has been observed that the probability of data landing within the area under the curve between 1 standard deviation to the left or right of the mean (0) is ~68%. It is ~95% likely to land between 2 standard deviations of the mean, and it is ~99.7% likely to land between 3 standard deviations of the mean. As I mentioned a z-score of -3.5 or 3.5 is equivallent to a 100% chance.

Remember that z-score is calculated using the following formula:

A quick review of rank statistics: Rank statistics is simply the categorization of data into percentiles. Assuming a maximum of 100%, we describe the amount of data as a fraction of 100. In rank statistics, 50% is the median percentile.

Another important thing to calculate when analysing trends in data is the covariance and correlation between the dependent and independent variables. The correlation describes the degree to which variables tend to move together. For instance, given the dependent variable, y, and the independent variable, x, correlation can be modeled using the following formula:

This will transform the correlation into a number between -1 and 1, 0 meaning no correlation, -1 meaning a 1:1 inverse correlation, and 1 meaning a 1:1 positive correlation.

Finally, we look at confidence intervals, which measure the uncertainty of a parameter. By taking multiple small samples, your values will always be different. The mean that you repeatedly sample will follow a normal distribution. The confidence intervals calculate the probability that the actual mean falls close to your measured number. Confidence intervals work in similar fashion to normal distribution, but they are not equivallent. Confidence intervals have a hard-coded lookup table similar to z-scores for something called the critical value (Zc). A critical value of 1.28 = 80% confidence level, 1.645 = 90%, 1.96 = 95%, 2.33 = 98% and 2.575 = 99%. A 90% confidence interval indicates that there is a 90% chance that the true mean falls within the range of repeated measurements that you sampled (give or take some margin of error). The formula for confidence levels is as follows:

where n is the number of samples that you took

Supervised Learning vs. Unsupervised Learning:

As mentioned in the introduction for this document, there are two types of learning that we will be looking at to train our prediction models with. The first kind is supervised learning. Types of supervised learning include classification and regression. For classification, we create a special attribute known as the class value, which will represent the thing that we are trying to predict. If we know the class value for a number of entries in what is called the “training set”, then we can use an algorithm to have our prediction model search for patterns within all of the attributes that are *not* the class value, and depending upon the strength of the correlation between certain attributes in the training set and the known class value, the prediction model will determine the probability of an unknown class value in the test set by simply looking for the same trends in the attributes of each instance/entry. Regression is similar, but it predicts continuous values as the output. The simplest way of describing regression is essentially finding the line of best fit and interpolating from that. Linear regression is the most common form of regression, as we just need to find the slope for the LOBF using everyone’s favorite formula: y = mx + b. We then do some relatively basic math to determine where the new interpolated value should fall relative to the LOBF. Note that there are other types of regression besides just linear. We can use quadratic, cubic, sinusoidal or other higher order functions at the risk of overfitting the data. Overfitting the data by making our predictive model *too* predictive means that it will not perform well in real world applications. For instance, imagine you give your predictive model a training set and you overfit the data by giving it a function that passes through every data point perfectly. This model will have essentially 100% accuracy depending upon the number of data points and standard deviation of the data being analysed. When pit against the test set however, if so much as minor changes exist, the model will likely perform poorly, because we’ve given it no predictive power. It is relying completely upon the function that we’ve given it, and that function was modelled too closely after our training set, not the broader trends that we’re looking for.

Unsupervised learning uses no class labels. Instead, we use a clustering algorithm to detect groups (clusters if you will) of data. It then tries again to determine trends in the attributes of the data, and then determine in which cluster it belongs and how closely it belongs to it. Something known as anomaly detection is the process of searching for outliers from the cluster e.g., when your bank calls you because you make a big purchase which is not usual for you to make.

Finally, we have semi-supervised learning, which typically consists of a small amount of labeled data, and a large amount of unlabeled data.

Training Set and Test Set:

As I mentioned in the previous section, we can optionally create two datasets: a training set and a test set. These two should not interact with each other or have influence over one another. Typically, the test set is taken as a sample of the entire dataset. Our training set is typically much larger than the training set. In supervised learning all class values should be known for the training set. However, for the test set, the class values should be removed, because the goal is to predict the values of the test set to determine the accuracy of the prediction model. The training set is just there to help the predictive model learn about the dataset’s trends. As a general rule of thumb, we tend to use the 70/30 rule, meaning that we take a sample size of 30% from the dataset to be used for the test set, and the remaining 70% is used for the training set.

Error Estimation:

A tool like Weka or other ML tools are capable of determining the “correctness” of their predictions using error estimation. There are different ways of calculating the error margin, but we will look at two: random sampling with repeated holdout and K-fold cross-validation. Random sampling with repeated holdout is when we first randomize our dataset and then split it into 2/3 training data and 1/3 test data. We then run our algorithm and record the evaluation score i.e., the error margin. We repeat this process until the error rate stabilizes i.e., it stops changing by a factor that is greater than some pre-determined threshold.

I’ll be brining up the constant variable ‘k’ a lot going forwards. The constant k is standard for a few different algorithms in ML. It is simply an arbitrary number that we choose depending upon the dataset and algorithm being used. In the case of K-fold cross-validation, k = the number of ‘folds’. The term fold is just a fancy way of saying groups. So for instance, 10-fold cross-validation would mean that we divide the dataset into 10 folds or groups. The way that k-fold cross-validation works is that we first randomize the order of the data, same as with random sampling with repeated holdouts, and then divide the dataset into the number of folds specified by k, and then we reserve one of those folds as the test set and the rest of the folds as the training set. We do this k-1 times, so if k = 10, then we train the model 9 times. For each fold we record the evaluation score and then take the average amongst them.

Accuracy:

Most ML tools will have a way of displaying what is known as a confusion matrix. A confusion matrix is a simple table that records the number of predictions that the model correctly identified, as well as the number of predictions it failed to predict. For a binary class value, there will be two rows: actually true (+) and actually false (-). There will also be two columns: predicted true (+) and predicted false (-). As you can probably figure out, the number scored for predicted true (+) will be the number of instances that the model thought would be true (note that the values don’t necessarily have to be true/false per se, it could be yes/no or hot/cold, or whatever you’d like). The predicted false (-) column records the number of instances that the model predicted to belong to the opposing class value. The actually true (+) row corresponds with the predicted true (+) column, meaning that the count that falls under both the aforementioned row and column will be the count of instances that were predicted to be true and were, in fact, actually true. The count that falls under the actually true (+) row and predicted false (-) column will be the number of instances that were actually true, but were incorrectly predicted as false. Likewise, the same logic applies for the actually false (-) row.

|  |  |  |
| --- | --- | --- |
|  | Predicted True (+) | Predicted False (-) |
| Actually True (+) | **A:** Actually True and Predicted True | **B:** Actually True but Predicted False |
| Actually False (-) | **C:** Actually False but Predicted True | **D:** Actually False and Predicted False |

The accuracy of the model can be determined by dividing the number of instances that were correctly classified by the total number of instances in the dataset. Looking at the labels A, B, C, and D in the table above, we can create a formula for accuracy: (A+D) / (A+B+C+D), where A and D are the cases that were correctly classified, and the sum of A, B, C, and D represents the total number of instances in the dataset.

The precision of the model is defined as A / (A+C). This represents the ratio of instances that were predicted true and that were actually true to to total number of instances that were predicted as true. Likewise, the specificity of the model is defined as D / (B+D), which represents the ratio of instances that were predicted to be false and were actually false to the total number of instances that were predicted false.

The recall or sensitivity are defined as A / (A+B). This represents the ratio of instances that were predicted as true and were actually true to the total number of actually true instances.

Nearest Neighbor Algorithm (kNN):

Now we get to our first algorithm that can be used for creating the predictive model! As mentioned earlier, we use the constant ‘k’ quite a bit in ML. Nearest Neighbor is an algorithm that you may or may not be familiar with, as it is used to compute a variety of things. I happened to recognize it, as it is a common algorithm used in scaling images. To summarize the algorithm as simply as possible, we basically just find the hypotenuse using Pythegoreans Theorum. Recall that Pythegorean’s Theorum for a 2-dimensional triangle is . For a 3-dimensional triangle, the formula does not change, except for the number of terms that we add to the sum of squares: . We simply add one additional term per dimension. The official term of the result for this calculation is the ‘Euclidian Distance’. Each ‘dimension’ is the ‘distance’ between the instance that we are trying to find the class value for and the ith instance in the training set. What do I mean by distance? I simply mean the mathmatical difference between the attribute in the instance that we are trying to find class value for and the same attribute value of the ith instance in the training set. Let’s say that we have a dataset that contains attributes for a person’s age, height, and weight. Assume the class value that we want to predict for is whether or not the person is employed (not a great example of a dataset, but it’s good enough for demonstrating how to calculate Euclidian Distance). First, we extract the instance that we want to predict the employed class for from the training set by either making a test set or just extracting the one instance. Then, we consider the 0th instance in the training set. We take the difference of age between this instance and the one we want to find employed status for. This is the ‘distance’ of age, or ‘a’ in our formula. We calculate distance for all other attributes between the two instances, then square each distance, sum them, and then take the square root of the sum. We now have the Euclidian Distance between those two instances. We repeat this process for each instance in the test set, keeping track of each Euclidian Distance. We then sort instances by Euclidian Distance and find the smallest distance. The smallest Euclidian Distance will indicate the instance with the most similarities on average. This is where ‘k’ comes into play. Let’s say k = 5. We take the top 5 smallest Euclidian Distances i.e. the top 5 most similar instances. We say that not employed = 0 and employed = 1 and then take the average of their class values and round the result to determine the class of the test instance. Seems confusing, but it’s actually quite simple. Due to the fact that we take the average of the top ‘k’ instances and round, kNN is considered to be a “majority wins” estimate algorithm.

Decision Trees:

The next algorithm we will look at are decision trees. A decision tree is a tree structure (a concept with which you are hopefully familiar). It is typically not a binary tree, and if it is, this is by coincidence only. Each node in a decision tree represents an attribute of the dataset and each branch represents a rule/decision. Each leaf represents an outcome i.e., the determined class of the test instance. Decision trees are one of the most popular ML algorithms and they can be used for both classification and regression. The downsides to decision trees are that they have a tendency to overfit the data. Ideally, we want to work with smaller trees since there are less things to compare. There are two types of decision tree algorithms: ID3 (Iterative Dichotomiser 3) and CART (Classification and Regression Trees). We will look at ID3. In order to determine which attribute to use as the root node, we must find the attribute with the highest “information gain”. Information gain can be thought of as the attribute that has the highest effect or influence on the dataset. In order to calculate information gain, we must first calculate the entropy of each attribute in the dataset. Entropy is defined as the amount of randomness or jitter. It will be a number between 0 and 1, where 0 means no randomness and 1 means complete randomness. We calculate entropy/impurity with the following formula: where S is the current attribute that we are calculating entropy. C is the set of classes e.g., C = {Hot, Cold}, and lowercase c represents each element belonging to that set. p(c) is the ratio of elements in class C to the number of elements in S. Assume we have an attribute, temperature, that has the nominal values {cool, mild, hot}. Here is the set C for the entire attribute column:

{hot, hot, hot, mild, cool, cool, cool, mild, cool, mild, mild, mild, hot, mild}. Since there are three possible values (cool, mild, and hot), we do the formula for each one before taking the sum. It gets more confusing though, because we need to split each attribute value into the two values for the binary class value that we are searching for. Let’s say that we are trying to predict the class value for whether or not children will play at a park, labeled ‘play’ for simplicity. For each of the three temperature values, we need to take the difference of the ones that have a class value of play and the ones that do not. Here is the same temperature feature column, with their corresponding class values:

Temperature: {hot, hot, hot, mild, cool, cool, cool, mild, cool, mild, mild, mild, hot, mild}

Play: {no , no , yes, yes, yes, no, yes, no, yes, yes, yes, yes, yes, no}

Here is the calculation:

As you can see, in the second to last step, we multiply the results of each value by the weight of the number of that nominal values (c) compared to the number of elements in the set (S). So since there are 4 instances that have a temperature feature of cool, we multiply by 4/14. Likewise, there are 6/14 instances that have a temperature value of mild, and 4/14 that have one of hot. Finally, we take the sum to get our entropy value of 0.8361.

Next, we must calculate the information gain. Recall that information gain represents the magnitude of influence that the attribute has on other attributes in the dataset. Information gain is calculated with the following formula:

where T is the subset created by splitting S by attribute A, p(t) is the proportion of the number of elements in t to the number of elements in S, and H(t) is the entropy of subset t that we calculated previously. Basically it’s just the difference between the entropy of the entire dataset and the entropy calculated for a specific attribute.

Once we have the information gain of each attribute in the dataset, we choose the one with the highest information gain at the root node. We repeat this process recursively until the algorithm is told to stop by complying with constraints such as the “minimum number of objects” which is the minimum number of nodes required to justify another split in the tree i.e., another recursion.

Random Forest:

As mentioned previously, decision trees can have a tendency to overfit the data. To alleviate this issue, if a decision tree is being used, we will typically also run the random forest algorithm to confirm our results or look for biases. As the name implies, a random forest is a collection of decision trees. We can choose how many trees we want and the depth (i.e., number of recursions) for each tree. In general, the more trees the more accurate the outcome, but there comes a point of deminishing returns, where adding more trees fails to yeild a higher degree of accuracy in the outcome. To give a brief about how random forests work, we essentially randomize the dataset for each tree in the forest using an initial seed value (this means that the randomization is only pseudo-random, since applying the same seed value yeilds the same results). Each decision tree predicts an outcome, and then we take the majority vote amongst each tree to determine the true classification value. The democratic nature of this algorithm is what helps alleviate the overfitting problem that we commonly find in decision trees.

Getting a bit more technical, we can discuss the different methods used for random forests – these being the bootstrap aggregation a.k.a. bagging, and boosting.

Clustering:

So far, we’ve covered kNN and decision trees, which are both algorithms that work for supervised learning models. Now we’ll take a look into an unsupervised algorithm, clustering. As mentioned prior, unsupervised learning means that we do not rely on a class value, we only have attributes to work off of. Clustering algorithms include k-Means, Mean-Shift, Density-Based Spatial Clustering of Application with Noise (DBSCAN), Expectation-Maximization, etc. We will be looking at the k-Means algorithm. K-Means requires us to take the square root on the data, therefore, it only works for numeric data. We will once again be calculating distance, similar to kNN. The way that it works is that we plot out our data into a 2-D scatter plot and randomly select a location to place what is called the centroid. We create k number of centroids. For each point in the plot, we first find its nearest centroid using Pythegorean’s Theorum to calculate distance. We then group the data into all the points who’s nearest centroid was centroid 1, the points who’s nearest centroid was centroid 2, etc... until centroid k. We recalculate each centroid’s location to the center of the cluster by taking the average distance of each group/cluster and repositioning it there. We repeat this process until each centroid stops moving. If we do this process for incremental values of k i.e., 2, 3, 4 … k and plot the average cluster distance to the centroid vs the number of clusters, we will get a graph that has a distinct/sharp “elbow” point, where the graph suddenly bends. The value of k at that elbow point represents the optimal value of k for our dataset.

Outlier Detection:

Another algorithm used for unsupervised learning is outlier detection, which I briefly mentioned earlier in the document. An outlier is any data point which deviates significantly from the mean of the set. If a specific instance contains an outlier, this is called a contextual outlier. If a subset of data objects collectively deviate from the dataset, this is referred to as collective outliers. Some methods of outlier detection include statistical methods, proximity-based methods (e.g., distance-based, density-based), and clustering-based.

One such method of determining outliers is to simply use Gaussian distribution and set limits for what is considered to be an outlier. For example, anything outside of 3 SDs to the right or left of the mean (100% - 99.7% = 0.3%) will be considered an outlier.

A density-based approach is called Local Outlier Factor (LOF). What makes LOF unique is that we compare outliers to their “local neighborhood” i.e., nearest cluster, rather than with the entire dataset.

Isolation Forests (ISF) are another method of outlier detection. The term “forest” refers to a collection of tree structures (in this case, a series of decision trees). We partition data by randomly selecting a feature from the dataset and then randomly splitting the value between the minimum and maximum values of the selected feature.

It is quite common to create what is referred to as an ensemble between LOF and ISF. Quite simply, we just take the results of the LOF’s detected outliers and the ISF’s detected outliers and only consider the instances which were classified as outliers by both algorithms as the true outliers. For instance, if LOF were to decide that an instance in the dataset *was* an outlier, but ISF determined that is was *not* an outlier, then ISF would overrule, and the instance would not be considered as an outlier. However, if both LOF and ISF agreed upon the instance being an outlier, we would consider it to be as such.

Association Rule:

Association rules are a unique method of data classification compared to the other approaches we’ve looked at. It is an unsupervised algorithm which is not suitable for numeric data, but rather categorical data. Associations are causal ties between two entities. Amazon’s ‘recommended for you’ tab uses association rules to observe causal ties between the purchasing of one item with other items. For example, customers who buy candles tend to also buy lighters. We use a format for describing this relationship, shown below:

A → B[Support, Confidence]

A is called the antecedent and B is called the consequent, because A is the given variable and B is the consequence/result of A. The support denotes the probability that A is given. For example, if total transaction in a store for the day is 100 and 20 of them contain bread, then the support will be 20%. The confidence is the probability that given A, B will be present. Reusing the previous example, if 9 of the 20 transactions containing A also contain B, then the confidence will be 9/20 = 0.45 \* 100 = 45%. We summarize this as A → B[20%, 45%]. Algorithms that can be used to calculate associations are the Apriori algorithm (this happens to be the best choice in basically all circumstances), the Elcat algorithm, and the F.P. Growth algorithm. Since Apriori is the best algorithm in nearly all cases, we will look at that one.

For Apriori, we first determine a “minimum support criterion”. This is the threshold that needs to be passed in order for any two items to be considered as having an association. The support criterion can be calculated as follows:

where Nt is the number of transaction in which X appears, and Tt is the total number of transactions. Consider the following list of transactions:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Transaction # | Onion | Potato | Burger | Milk | Pepsi |
| T1 | 1 | 1 | 1 | 0 | 0 |
| T2 | 0 | 1 | 1 | 1 | 0 |
| T3 | 0 | 0 | 0 | 1 | 1 |
| T4 | 1 | 1 | 0 | 1 | 0 |
| T5 | 1 | 1 | 1 | 0 | 1 |
| T6 | 1 | 1 | 1 | 1 | 0 |

Here is the list of all items that appear in the set of transactions and their frequencies:

{[onion, 4], [potato, 5], [burger, 4], [milk, 4], [pepsi, 2]}

Let’s assume that we set a support threshold of 50%. 50% of 6 transaction is 3, so items that have a transaction frequency less than that are not considered i.e., pepsi. Next, we calculate the support for each item in the set that is being considered:

Support(onion) = 4/6 = 0.6667

Support(potato) = 5/6 = 0.8333

Support(burger) = 4/6 = 0.6667

Support(milk) = 4/6 = 0.6667

Now, we begin to join items which have passed the threshold of 50% so that we have every possible combination of item. The count is now based upon the number of transactions that contain *both* items. This is called the 2-item set, and we are going to repeat this up until k-item sets:

{[{onion, potato}, 4], [{onion, burger}, 3], [{onion, milk}, 2], [{potato, burger}, 4], [{potato, milk}, 3], [{burger, milk}, 2]}

As you can see, we’re now looking at transaction which contain 2 items at a time. Once again, we calculate support for each pair of items:

Support(onion, potato) = 4/6 = 0.6667

Support(onion, burger) = 3/6 = 0.5

Support(onion, milk) = 2/6 = 0.3333

Support(potato, burger) = 4/6 = 0.6667

Support(potato, milk) = 3/6 = 0.5

Support(burger, milk) = 2/6 = 0.3333

Now we remove onion → burger and burger → milk since they did not meet the 50% cutoff. Assuming k = 3, we can do this one more time. Our 3-item set looks like so:

{[{onion, potato, burger}, 3}], [{potato, burger, milk}, 2}]}

And the supports:

Support(onion, potato, burger) = 3/6 = 0.5

Support(potato, burger, milk) = 2/6 = 0.3333

And there we have it! Onion, potato, burger have an association! We can calculate the confidence that the antecedent X contains the consequent Y by dividing the support of a union between the two of them over the support of X. Here is the formula:

Here is the confidence that milk exists given onion and potato:

Conf({onion, potato} → milk) = Support({onion, potato} U milk) / Support({onion, potato})

= Support({onion, potato, milk}) / Support({onion, potato})

= (3/6) / (4/6)

= 0.0208

This is not a percentage, it is a normalized value between 0 and 1, where 1 means 100% confidence and 0 means no confidence.

You might also want to calculate something called lift. Lift measures the ratio between the frequency that X and Y happen together vs how often they appear by themselves. It is defined as:

This will give a number within the range 0 to infinity. 0 indicates that there is a 1:1 negative correlation, 1 indicates no correlation, and anything above 1 indicates a positive correlation.

Regression:

I briefly discussed regression, which is an unsupervised learning algorithm that essentially uses the line of best fit to interpolate or extrapolate data. As mentioned, it is common that we use linear regression, as this is the simplest, but it is a bit more complex than that in reality. First, lets consider the two types of relationships that we can have between attributes. The first type of relationship is known as a deterministic or functional relationship. In a deterministic relationship, the x-axis is the independent variable, and the y-axis is the dependent variable. This means that the outcome of y depends upon the input, x. To be more precise, in a deterministic relationship, the outcome of y will always be the same given a specific value of x. Take the relationship between temperatures in Celsius and in Fahrenheit, for example. If Celsius is our x-axis, and Fahrenheit is our y-axis, the value of y where x = 10 will always be 50. The second type of relationship is known as a statistical relationship. In this type of relationship, y is affected by x, but not completely determined by x. For example, as height increases, you would expect weight to increase, but we cannot perfectly predict what weight will be given someones height. Regression is useful for modeling statistical relationships as if they were deterministic relationships.

The first algorithm that we will analyse is Simple Linear Regression. In SLR, the independent variable (x) is called the predictor, and the dependent variable (y) is called the response. Although we are used to the formula y = mx + b to model linear equations, data scientists had to be annoying about it and remodel the formula as where theta0 and theta1 are the coefficients. The fancy term that we use for finding the LOBF is “Univariate linear regression”. In order to choose the optimal values for theta0 and theta1, we use a cost function. The cost function calculates the total error between the predicted values i.e., the LOBF and the actual values. We continue to change the LOBF until we find the minimum error. We use a method known as the “least squares criterion” to find the overall prediction error. As you may know, absolute error (AE) is the actual value – the predicted value. The squared prediction error is just AE squared. To compute overall squared prediction error, we just take the sum of squared prediction errors for each point in the dataset. In other words, if Yi is the actual value, and Yi’ is the predicted value, the formula for overall squared prediction error will be

In order to determine if the regression line is a good fit i.e. not overfitting or underfitting, we can calculate the “Coefficient of determination” a.k.a. the residual, a.k.a. where ‘r’ is the correlation coefficient. The coefficient of determination will be a normalized number between 0 and 1 which represents how “close” your data is to the line. If the regression line passes through each point in the dataset, will be = 1.

Let’s now discuss Multivariate Regression. This is a technique which estimates a single regression model with more than one outcome variable. Instead of just using y as the response, we may now have multiple responses. For example, a doctor collects data on cholesterol, blood pressure and weight and wants to determine the relationship between these and the number of ounces of red meat, fish, dairy products, and chocolate are consumed per week.

Logistic Regression models a relationship between the predictor variable and a categorical response variable. This helps us estimate the probability of falling into a certain level of the categorical response given a set of predictors.

Business Intelligence:

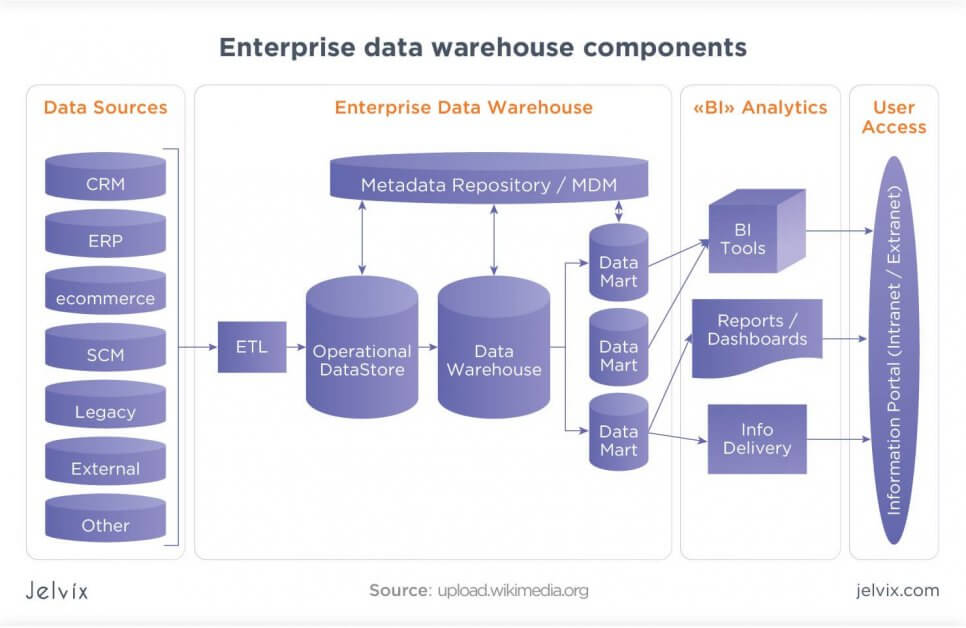
Let’s talk about business intelligence as a whole, and some of the terminology or more abstract concepts that you may encounter while working within the field. Business Intelligence systems have been around for almost 20 years. They try to summarize large datasets and understand what is happening. They attempt to detect trends to help the staff make informed decisions. You may be wondering what the difference is between business intelligence and data analytics. The primary difference beteween these two is that business intelligence describes past occurences using historical data, whereas data analytics attempt to predict future trends. There are three different types of decisions that business intellegence systems can help staff make:

* **Strategic:** Using information to make better decisions
* **Administrative:** Making short-term decisions that occur frequently
* **Operational:** Helps identify the processes that require optimization

Business Intelligence systems are comprised of four primary components:

* **Data Warehouses:** A large store of data that is accumulated from different sources within a company
* **Business Analytics tools:** Manipulating and analysing data
* **Business Performance tools:** Monitoring and analysing performance
* **Visualization:** Portals, dashboards, scorecards, etc.

Let’s look into data warehouses further, as these are obviously quite important for business intelligence systems. Data warehouses pull data from a corporations inventory, sales, etc. These are typically updated on a daily basis. The process of gathering data is referred to as Extract, Transform, Load (ETL). Extraction pulls data from its raw source. Transform manipulates the data to convert it into the desired format. Load stores the data in the final target (the data warehouse or data mart). If the data won’t fit into memory, then it must first be stored in a DB and transformed later. In this case, the process is referred to as Extract, Load, Transform (ELT), since the data must be loaded before it can be transformed.

  
 A Data Mart (depicted in the image above, to the right of the data warehouse) is essentially a minature data warehoue, which typically addresses one portion of the warehouse. For example, you may have a mart for sales, marketing, finance, etc. Data marts are extracted from the data warehouse and can store summarized data instead of raw data.

It may seem obvious to compare data warehouses with databases, but these two are not identical. Data warehouses are not normalized as databases are. Due to this, they use less tables and don’t exclude redundant data. They pull data from various sources for reporting and analysis and try to do this as quickly as possible. A database, on the other hand, has normalized data which are sorted into columns, rows, and tables. Data is typically organized to make sure there is no redundant data, and normalized data allows for more complex queries so that a single read might combine data from multiple tables.

Related to data warehouses are OnLine Transaction Processing (OLTP) systems and OnLine Analytical Processing (OLAP) systems. An example of an OLTP system are the payment processors found in stores. These are continuously running and accepting input from the end user. They ensure quick transactions and maintain data integrity. They primarily perform INSERT, UPDATE, and DELETE operations. OLAP, on the other hand, operate long-running queries and look for patterns in the data. These are systems that generate business reports and mine the data. For example, a company might print off a weekly sales report, which will use an OLAP system to pull from the data warehouse and provide a summary.

|  |  |
| --- | --- |
| **OLAP** | **OLTP** |
| Works on historical data | Works on current data |
| Helps analyse the business | Helps run the business |
| Provides summarized data | Provides raw data |
| Uses long-running queries to refresh summaries | Focuses on fast & secure queries |
| Used by specialized users for decision support | Used by normal company staff |

Decision Making:

In decision making, it is important to understand what is happening so that we can do something about it. There are several types of Decision Support Systems (DSS) that can help us solve decisions. There are 3 primary types of decisions that a DSS can help us solve:

* **Structured decisions:** Choosing whether to buy a new product or service, what inventory levels are required, etc.
* **Semi-structured decisions:** Settings budgets, pricing, etc.
* **Unstructured decisions:** No clear solutions e.g., advertising, designs, etc.

There are various types of DSS:

* **Data driven:** Process numerical data from the data warehouse. Produces dashboards and scorecards
* **Document driven:** Works on documents, videos, transripts, media
* **Knowledge driven:** Uses knowledge and past experiences to support decision making e.g. a neural network
* **Model driven:** Uses models and simulations to support decision making
* **Communication driven:** Group support systems for collaboration e.g. Facebook, Skype, etc.

Business Performance Tools:

There are several types of tools that can be used to measure performance. Here are a few of them:

* **Performance appraisals:** A systematic and periodic process of measuring one’s performance against the job and company requirements
* **360-degree feedback:** Confidential anonymous feedback from people who work around them
* **Key Performance Indicators (KPI):** Set of indicators to measure data against a goal or success metric
* **Personal Development Plans (PDP):** An action plan. Putting goals into achievable steps.
* **Reward and recognition programs:** Celebrating employee success

Characteristics of Big Data (Five Vs):

There are 5 ‘v’ words that can help summarize the various characteristics of big data that can be analysed. Here is a list and their definitions:

* **Volume:** Refers to the vast amount of data that is generated every second/minute/hour/day in the digitized world. Examples include GPS sensors, accelerometers, gyroscopes, etc. Also social media such as Facebook, Twitter, etc.
* **Velocity:** The speed at which data is being generated and the pace at which data moves from one point to the next
* **Variety:** The ever-increasing forms that data can come in. This brings challenges in terms of data integration, transformation, processing, and storage.
* **Veracity:** The quality of data, which can vary greatly.
* **Value:** The usefulness of data for an enterprise. Value and time are inversly related. The longer it takes for data to be turned into meaninful information, the less value it has for a business.

Big Data Analytics Lifecycle:

1. Business case evaluation
2. Data identification
3. Data acquisition and filtering
4. Data extraction
5. Data validation and cleansing
6. Data aggregation and representation
7. Data analysis
8. Data visualization
9. Utilization of analysis results

Big Data Storage Concepts:

* **Clusters:** A cluster is a tightly coupled collection of servers or nodes. These servers usually have the same hardware specifications and are connected together via a network to function as a single unit. Each node in the cluster has its own dedicated resources, such as memory, processor(s), and hard drive(s). A cluster can execute a task by splitting it into small pieces and distributing their execution onto different computers that belong to the cluster.
* **Distributed file systems:** A file system is a method of storing and organizing data on a storage device, such as flash drives, DVDs and hard drives. A file system provides a logical view of the data stored on the storage device and presents it as a tree structure of directories and files. A distributed file system is a file system that can store large files spread across the nodes of a cluster. To the client, files appear to be local; however, this is only a logical view, as physically, the files are distributed throughout the cluster. This local view is presented via the distributed file system and it enables the files to be accessed from multiple locations.
* **Sharding:** Process of horizontally partitioning a large dataset into a collection of smaller, more manageable dataset called shards. The shards are distributed across multiple nodes, where a node is a server of a machine. Each shard is stored on a separate node and each node is responsible for only the data stored on it. Each shard shares the same schema, and all shards collectively represent the complete dataset. Sharding allows the distribution of processing loads across multiple nodes to achieve horizontal scalability. Horizontal scaling is a method for increasing a system’s capacity by adding similar or higher capacity resources alongside existing resources. Since each node is responsible for only a part of the whole dataset, read/write times are greatly improved.
* **Replication:** Stores multiple copies of a dataset, known as replicas, on multiple nodes. Replication provides scalability and availability due to the fact that the same data is replicated on various nodes. Fault tolerance is achieved since data redundancy ensures that data is not lost when an individual node fails. There are two different methods that are used to implement replication: Master-slave and peer-to-peer.
  + **Master-slave replication:** Nodes are arranged in a master-slave configuration and all data is written to a master node. Once saved, the data is replicated over to multiple slave nodes. All external write requests, including insert, update, and delete, occur on the master node, whereas read requests can be fulfilled by any slave node. Master-slave replication is ideal for read-intensive loads rather than write-intensive loads, since growing read demands can be managed by horizontal scaling to add more slave nodes. Writes are consistent, as all writes are coordinated by the master node. The implication is that write performance will suffer as the amount of writes increases. If the master node fails, reads are still possible via any of the slave nodes. A slave node can be configured as a backup node for the master. In the event that the master node fails, writes are not supported until a master node is reestablished. The master node is either resurrected from a backup of the master node, or a new master node is chosen from the slave nodes.
  + **Peer-to-Peer:** With peer-to-peer replication, all nodes operate at the same level. Each node (peer) is equally capable of handling reads and writes. Each write is copied to all peers.
* **ACID:** A database design principle related to transaction management. ACID is an acronym:
  + **Atomicity:** Ensures that all operations will always succeed or fail completely (no partial transactions)
  + **Consistency:** Ensures that the database will always remain in a consistent state by ensuring that only data which conforms to the constraints of the database schema can be written to the database.
  + **Isolation:** Ensures that the results of a transaction are not visible to other operations until it is complete.
  + **Durability:** Ensures that the results of an operation are permanent. In other words, once a transaction has been committed, it cannot be rolled back. This is irrespective of any system failure.

Trending Topics:

Sentiment analysis is contextual mining of text which identifies and extracts subjective information in source material, and helping a business understand the social sentiment of their brand, product, or service, while monitoring online conversations. This appoach falls under Natural Language Processing (NLP). Sentiment analysis is sometimes referred to as opinion extraction, opinion mining, sentiment mining, and subjectivity analysis. In essence, sentiment analysis studies public reception to something e.g., politics. Analysis of sentiments can be accomplished using ML (supervised/unsupervised), using lexical analysis e.g. finding keywords in online posts, or discourse analysis.

Link Analysis:

Link analysis is a data analysis technique used to evaluate relationships (connections) between nodes. Relationships may be identified among various types of nodes (objects), including organizations, people, and transactions. This is often used for investigating criminal activity (fraud detection, counterterrorism, and intelligence), computer security analysis, search engine optimization (SEO), market research, medical research, and art. We load a claim i.e., a relationship, and then we find other claims which are very similar. Finally, we combine matches to detect outliers and discover suspicious activity.