ACTUARIALINTELLIGENCE: DOMAIN DRIVEN DESIGN FOR RISK SOLUTIONS. ADVICE FOR ACTUARIAL AI TEAMS

RAJAH IYER

Abstract. We present herein the concept of an Atomic Domain Object, useful in understanding the concept of Domain centric logical modelling via the use of an example involving the Annuity instrument.

Introduction

The current state of Analytics will trump common intuition but understanding the mechanism by which these computer found correlations $\rightarrow$ intuitions come to be, will create a need for an intuition on how best to improve on such *intuitions* and bring into fruition better methods capable of observing patterns in the noisy-data. Historically machine learning algorithms have and continue still to work by noticing correlations in data. The nature of the data is irrelevant, for instance, given a data-warehouse consisting of individual purchase history (of all supermarket chains collectively); the current machine learning techniques are capable of predicting purchase correlations/likelihoods within the data, so for instance it is likely to point out that those purchasing beer are likely to purchase diapers as well; this knowledge would give the analyst information for advising purposes. Further to this, the analyst can now strategize and plan for how and what new data to collect however this can only be achieved by having insights into the data mining techniques themselves and an insight into what data needs collection for the efficacy of some desired goal. To highlight this further; purchase history data is naturally only going to bring to surface purchase correlations such as the one noticed above, and intelligent amendments can be made by inclusion. So for instance if one wished to know if other hidden correlations prevail; perhaps those specific to season and time of day, the analyst could perform the task with a request for more information surrounding those already gathered within the warehouse and run the algorithms against this new data to see if other correlations do in fact exist.
Medical analysts (those proficient in statistical machine learning techniques and medicine) constantly and based on hypothesis aim to find hidden correlations in the data they collect. The analysis engine can never know the objective and behave in accordance, it is the analyst's job to understand the hypothesis, the engine and additionally needs to have the prowess to know the structure and dimensions of the data required to come to a related conclusion as one is likely not to have

Key words and phrases. Domain Object, Domain Driven Design, VaR (Value at Risk), Annuity, Z-Spread, Spot Rate, Forward Rate.

'all' the data. Trivially a hypothesis would be: fast foods are the major contributor for diabetes, and to assert this, one would naturally gather data pertaining to fast foods, their contents, and what percentage of the populous has take-out and what percentage of the time.. A simple hypothesis (such as the chi-squared) test would suffice for conclusion, however given that; given sufficient data pertaining to peoples eating habits in a data-warehouse would realise the same conclusion, the analyst can now strategically gather intuitive data homogeneous to drivers for instance having one or more medical conditions and find correlations thereof for ascertaining whether accidents were more or less prevalent amongst this group and steadily navigate toward which diseases has this mysterious influence, thus paving the way for advice for setting insurance premiums.

The take-away point here is that the analytics engines are not replacing the Analyst, instead these are changing the nature of the analysis needed by the analyst to a more engaging one requiring enlightened intuition best achieved by understanding the nature of machine learning algorithms along with data orchestration mechanisms and AI. The easiest analogy for this is that: Google search engines make use of a vast array of data-mining algorithms; and simply by understanding this the user can structure the search query in a manner yielding the best possible array of matches. The user here is equivalent to the analyst, and to suggest that the behind the scenes calculations will replace the Google-searcher is the same as the suggestion that analytics will replace at present; the analyst; which is clearly not the situation.

Suppose one wished to have a robust investigation into customer satisfaction at distributed customer sites. Suppose further that aside from video footage the customer were not in possession of any sentiment data, then this is one example of a situation in which one can leverage existing data(in this case, the video footage data) in the way of producing valuable sentiment indicators necessary for this analysis. Using Azure Face API, one can collect sentiment data for time series analysis, for instance by, building APIs built to subscribe to a sentiment such as unhappiness within a threshold, and record this value along with a date-stamp and optionally some metadata such as location of site etc. By doing so, the Data Scientist now has staged data to work with. This is not to suggest that one can dispense with the need for additional valuable sentiment data such as collection of customer satisfaction data directly. This is mainly to draw attention to the possibility of producing such data in circumstances where collection of the needed data maybe impossible. There is a lot to unpack in this arena, so to continue it suffices to say that such possibilities are ever-present and it is the job of the apt Data Scientist to be cognisant of this. This being said, consider the position of the data scientist tasked with this within a single processing platform such as Azure DataBricks, the collective task of streaming, processing and analysing the data will be challenging. The reasons for this include and are not limited to, the means by which(using a vast array of staging patterns) data is staged. This is manual and highly time consuming depending of course on the amount of data and the nature of analysis required. Thus, though it is possible in Azure to automate certain scripts and abstract this process out, In situations where communication with and processing via the use of APIs is

required, the recommended means by which this is achieved is via Azure Functions and APIs.
Combining the ability of efficiently being able to stream event specific data via Functions and APIs capable of leveraging Cognitive Services along with having the ability to effectively read data from almost all source types and stage this in a conducive manner will make for very effective analysis turnaround times with reliable results, as opposed to those open to human error(It is assumed here that all Azure Functions and APIs have gone through a full development life-cycle with testing and sign-off by qualified members with the appropriate authority).

1. The Theory.

The formula for an Annuity follows as:

$$Xa_{\overline{n}|}^{@j} = XV_{@i_1} + XV_{@i_2}^2 + .. + XV_{@i_n}^n \tag{1}$$

The concept of an annuity is composed of a Power Series Sum of Discount Factors, with each discount factor applicable to an interest rate. Assuming for the present moment that the interest rates are constant, then the natural concerns of this instrument are, estimating its present value given an interest rate, estimating the interest rate given the present value and the instalment value X, this involves interpolation(also a Domain level Concept), and finally estimating X given PV, i.
Supposing one wished to write a program capable of performing tasks specific to each concern above in the most minimal and re-useable manner possible, then one will find it best to abstract out the Discount Factor and Interpolation concepts and have the *Annuity object* call on these as needed.
The Zero-volatility spread (Z-spread) is the constant spread (j) that makes the price of a security equal to the present value of its cash flows when added to the yield at each point on the spot rate Treasury curve where cash flow is received. In such calculations again, having interpolation abstracted will be an advantage, as can be seen when considering its wider application to calculate say volatility in *option pricing*.

The price of a short term stock option in accordance with Black and Scholes follows as:

$$C(t, S_t, K, T) = S_t Q^S(S_T > K) - e^{-r\tau} K Q(S_T > K) \tag{2}$$

commonly written:

$$C = SN(d_1) - N(d_2)Ke^{-rt} \tag{3}$$

Where

$$d_1 = \frac{ln(S/K) + (r + (annualizedvolatility)^2/2) * T}{annualizedvolatility * (T^{0.5})} \tag{4}$$

The formula for d_2 is: $d1 - (annualizedvolatility) * (T^{0.5})$.

For all exposed under the same *annualized volatility*, the expectation of the price of the option follows as: $E[C] = \sum_{\forall t} f_{D_i}(t)C(t)$

It is clear that Interpolation needs abstraction and in a manner capable of passing in any arbitrary function for interpolation purposes. Though the Annuity object composes of the Interpolation, it too is *Atomic* in nature as it exposes a single responsibility domain capability. It is not entangled with the responsibility of transforming, reading/writing or any other concern. In a less financial context, other domain capabilities can be a call to an API responsible for encryption alone. To draw a clear picture:

Description: Atomic Domain Capability/Object
an *Atomic Domain Capability/Object*, it is a block of code housing logic for a single responsibility Domain Capability.

```
public static class Interpolation
 {
     private static decimal i1, i2 = 0;
     private static decimal f = 0;
     private static decimal previousValue = 0;

     /// <summary>
     /// Method to interpolate to a stipulated degree of accuracy the value of a functional.
     /// </summary>
     /// <param name="functional">Function as Func</param>
     /// <param name="testValue1">Interpolation lowerbound</param>
     /// <param name="testValue2">Interpolation upperbound</param>
     /// <param name="interpolationValue"></param>
     /// <returns></returns>
     public static decimal Interpolate(Func<decimal, decimal> functional,
         decimal testValue1, decimal testValue2, decimal interpolationValue)
     {
         decimal tempI = 0;
         i1 = testValue1;
         i2 = testValue2;
         f = interpolationValue;
         bool _continue = true;

         while (_continue)
         {
             tempI = i2;
             previousValue = i2;
             if (functional(i1) == functional(i2)) { break; }
             i2 = nextValue(functional(i1), functional(i2), i1, i2, out _continue);
             i1 = tempI;
             Console.WriteLine(i2);
         }
         return i2;

     }
```

```
        private static decimal nextValue(decimal f1, decimal f2,
            decimal i1, decimal i2, out bool _continue)
        {
            _continue = Threshold.Equals((i1 + (i2 - i1) * ((f - f1) / (f2 - f1))), previousValue);
            return (i1 + (i2 - i1) * ((f - f1) / (f2 - f1)));
        }

    }

    internal static class Threshold
    {
        private static decimal accuracyThreshold = 0.000000001m;
        public static bool Equals(decimal currentValue, decimal previousValue)
        {
            bool result = Math.Abs(currentValue - previousValue) < accuracyThreshold ? false : true;
            return result;
        }
    }
```

This type of atomic formulation, if done correctly, can have the effect of being quite versatile and re-usable. Suppose for instance suppose that we wished to reuse the Annuity object(See Appendix) in Annuity Equations, such as:

$$Xa_{\overline{n}|}^{@j} - Ya_{\overline{m}|}^{@i} \tag{5}$$

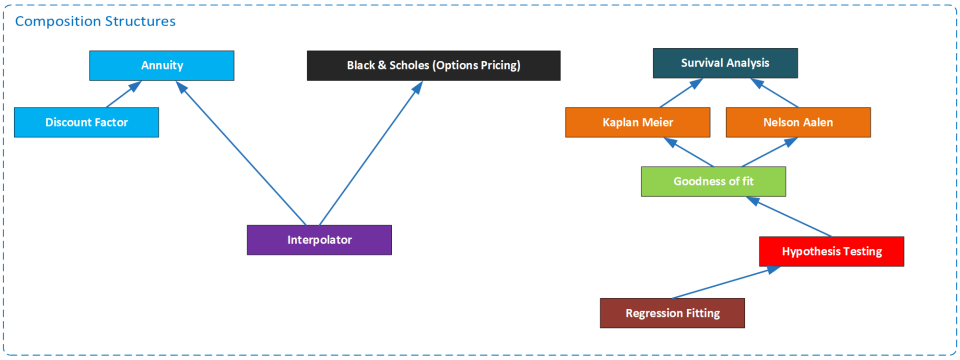
so as to obtain the Z-Spread, the following will suffice to achieve this:

```
    public class AnnuityEquations
    {
        private readonly Annuity annuity1;
        private readonly Annuity annuity2;
    public AnnuityEquations(Annuity annuity1,
        Annuity annuity2)
        {
            this.annuity1 = annuity1;
            this.annuity2 = annuity2;
        }
        public decimal ZSpreadOfAnnuityModel(decimal zSpread)
        {
            return 7.14m * annuity1.GetZSpreadPV(zSpread)
                - 2 * annuity2.GetZSpreadPV(zSpread);
        }
    }
```

(Other examples involving Annuity can be seen within the Appendix section.)[1]

[1]The problem with expecting libraries to behave in an efficiently conducive and reusable manner is that in most occasions these are built functionally and become difficult to customize and manage, creating the need for adding possibly unnecessary wrapper logic.

2. Abstracting the Domain



Where it makes sense, we avoid APIs consuming APIs when it comes to implementing one such Domain Object. So for instance, as the Annuity object makes use of Interpolation, it is unnecessary to use the interpolation API, as consuming it directly does not violate the single domain capability principle, and using the API would introduce latency and performance issues.

Other domain concepts such as Survival Analysis can be abstracted atomically as well, and one should always look for such opportunities with all Financial Concepts.
The standard approach to Survival Analysis involves Kaplan-Meier and Nelson-Aalen techniques, and these can be used effectively to obtain survival probabilities $S_{D_i}(t)$ per demographic type-set D_i. See for example 4.
As we aim more to ascertain the actual $PDF'_{D_i}s$ and less to model these, this is the preferred approach, (See 2 and also 1).
Making use of:

$$S(t) = \Pi_{i:t_i \leq t}(1 - \frac{d_i}{n_i}) \tag{6}$$

One can estimate $\lambda_{D_i}(t)$ by making use of the equation:

$$S(t) = e^{-\int \lambda_{D_i}(t)dt} \tag{7}$$

Which in discrete steps evaluates to:

$$S(t) = e^{-\sum \lambda_{D_i}(t)\Delta t} \tag{8}$$

Thus allowing for hazard estimations at each step. Keeping in mind:

$$\lambda_{D_i}(t) = \frac{f(t)}{S(t)} \tag{9}$$

We arrive at the following:

$$\begin{bmatrix} f_{D_1}(t_p) = S_{D_1}(t_p)\lambda_{D_1}(t_p) \\ f_{D_2}(t_p) = S_{D_2}(t_p)\lambda_{D_2}(t_p) \\ \dots \\ f_{D_n}(t_p) = S_{D_n}(t_p)\lambda_{D_n}(t_p) \end{bmatrix}$$

The above serves as a fairly accurate estimate for the nature of all PDF_{D_i}.

Tested and Published frameworks capable of automating all such estimations, that

equally integrate well with any number of data host systems are available. The interested reader can see for instance3.
The following image shows a sample Kaplan-Meier estimate based output over ten months for a specific set of exposed to risk members of a life-instrument.
In addition to the calculated points being obtained, one can request comparisons of fit's and a goodness-of-fit test used by artificially intelligent algorithms to select the best fit. Choosing the best fit ensures the estimation of a fair premium.

Concepts such as Goodness of Fit should additionally be abstracted out and exposed, as these are fundamental concepts specific to the Financial Domain.
Separation of concerns is in no way a new concept and we are employing it here so as to be able to expose these as functions capable of being docked and exposed on top of/within Kubernetes and the like. Calling applications will typically be n-Layered and make use of these domain functionalities and write the results.

The natural advantage here being that the architecture is built in a manner separating domain concerns and built in a manner capable of centralized access and scalability. (For a fully architected financial trade platform employing the above concepts, see, `https://github.com/ActuarialIntelligence/KDBADXandLogAnalytics`)

3. VaR (Value at Risk)

Under the Monte Carlo method, Value at Risk is calculated by randomly creating a number of scenarios for future rates using non-linear pricing models to estimate the change in value for each scenario, and then calculating the VaR according to the worst losses.

The steps required of such as estimation would require the the generation of future rates, and based on these the present valuing of an array of financial instruments within scope. Given that the latter portions have been abstracted out as per the the initial portion of this article, one need only iterate over simulated interest rates given an underlying distribution assumption.

4. Appendix

C#:

```
public class ZSpread // This is an arbitrary implementation of the Z-Spread consumer class.
{
    private ListTermCashflowSet cashFlowSet;
    int days;
    private decimal nominal;
    private decimal spread;

    public decimal Spread()
    {
        return spread == 0 ? CalculateZspread() : spread;
    }

    public ZSpread(ListTermCashflowSet cashFlowSet, decimal nominal)
    {
        this.cashFlowSet = cashFlowSet;
        this.nominal = nominal;
        days = cashFlowSet.termType == Term.YearlyEffective ? 365 :
            cashFlowSet.termType == Term.MonthlyEffective ? 30 : 0;
    }
    /// <summary>
    /// Obtains Z-Spread value via interpolation and returns the value.
    /// </summary>
    /// <returns></returns>
    public decimal CalculateZspread()
    {
        Annuity annuity = new Annuity(cashFlowSet, days);
        var result = Interpolation.Interpolate(annuity.GetZSpreadPV, 0.01m, 0.09m, nominal);
        spread = result;
        return result;
    }

}
```

```
public class Annuity // This is an arbitrary implementation of theconsumer class.
{
    private readonly ListTermCashflowSet cashFlowSet;
    private int days;
    public Annuity(ListTermCashflowSet cashFlowSet, int days)
    {
        this.cashFlowSet = cashFlowSet;
        this.days = days;
    }

    public decimal GetZSpreadPV(decimal zSpread)
    {
        var total = 0m;
        DateTime anchorDate = cashFlowSet.AnchorDate;
        foreach (var cashFlow in cashFlowSet.CashflowSet)
        {
            total += cashFlow.cashflow * discountFactor(cashFlow.spotYield.Yield + zSpread,
                cashFlowSet.DifferenceInUnitsBetweenDates(anchorDate, cashFlow.date, days));
        }
        return total;
    }

    public decimal GetPV()
    {
        var total = 0m;
        DateTime anchorDate = cashFlowSet.AnchorDate;
        foreach (var cashFlow in cashFlowSet.CashflowSet)
        {
            total += cashFlow.cashflow * discountFactor(cashFlow.spotYield.Yield,
                cashFlowSet.DifferenceInUnitsBetweenDates(anchorDate, cashFlow.date, days));
        }
        return total;
    }

    private decimal discountFactor(decimal yield, decimal term)
    {
        var d = (1 / (1 + yield));
        var dp = Math.Pow((double)d, (double)term);
        return (decimal)dp;
    }
}
```

References

[1] Kaplan EK, Meier P. Nonparametric estimation from incomplete observations. J Am Stat Assoc. 1958;53(282):457–481.

[2] Brandon George, Samantha Seals, Inmaculada Aban. "Survival analysis and regression models". https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4111957/

[3] GitHub. (2018 June 28). ActuarialIntelligence/Base. https://github.com/ActuarialIntelligence/Base

[4] Nelson DE Pril. "THE AGGREGATE CLAIMS DISTRIBUTION IN THE INDIVIDUAL MODEL WITH ARBITRARY POSITIVE CLAIMS". http://www.actuaries.org/LIBRARY/ASTIN/vol19no1/9.pdf