Automating the tagging of content for reuse

Authors:

Arnaud Letondor & Ian O’Keeffe - Learnovate Centre

Core Research Report

March 18

Table of Contents

1. Purpose and scope

For many years, Machine Learning (ML) has been investigated by Artificial Intelligence (AI) experts and enthusiasts and has been applied in fields such as robotics and computer vision. More recently, ML has seen much wider interest in areas as diverse as banking and fraud detection, medical diagnosis and language processing, making it one of the major trends in the 2010s.

Machine Learning is a specific application of the broader concept of Artificial Intelligence (AI), based around the idea of "learning" from data, using many examples (data) to answer specific questions. Although the origin of the term "machine learning" goes back to Arthur Samuel in 1959, ML has become more and more popular thanks to recent advances that have been made in the field. These advances in combination with the growth in the availability of computing power have prompted a growth in interest in the field and its application to solve new problems that would have previously been considered out of reach. More recently we have seen ML move from the lab into production environments and beyond that into our popular culture. Many newspaper column inches have been devoted to lauding the utopian future that ML will afford us while at the same time struggling to understand its social implications. Another significant recent trend in ML has been the increasing ease with which the technology can be accessed. With the availability of cloud-based infrastructure and APIs, open source ML frameworks as well as other libraries, access to such advanced technology has never been more straightforward. Despite this, ease of access is not the same as ease of use! We have all the tools at our disposal that we need to utilise ML but how practical is it to use those tools to solve real world problems that we might face in to the products and solutions that we make. How deep does our knowledge of ML and AI theories and techniques need to be before we can take our initial steps into the future that the technology evangelists have promised? Will it always be the case that only the multinational tech giants with their teams of PhD graduates can wield the power or can smaller organisations hope to apply it in ways that make meaningful impacts for their customers and ultimately for their organisations.

1.1. Content Tagging

A significant impediment to the application of enabling technologies such as content personalisation is the cost and effort on the part of the EdTech vendor to process content so that it can be readily consumed by the personalisation platform, or whichever technology is in question. Part of this process often involves the creation of metadata to describe the source content, allowing the content to be discovered and manipulated by the system as required. Currently this tends to be a manual process that requires a level of expertise, for example in instructional design, to correctly identify appropriate metadata tags. The quality of this tagging task is paramount as poor-quality classification/tagging of resources will have a direct and significant impact on the quality of the end-product and the experience of learners.

The manual generation of metadata severely limits the application of many different technologies that have the potential to enhance the experiences of learners and knowledge workers in a range of different contexts. A specific example of this limitation is the Almanac technology demonstrator that was developed by the Learnovate Centre several years ago. Almanac is a dynamic and adaptive content composition platform designed to support both formal and non-formal learning scenarios. As a content driven adaptive learning tool, Almanac needs to be aware of the content that is available to the system. As part of the design of Almanac consideration was given to the need to generate information that describes the available content and the expense and complexity involved, which lead to an architecture that minimised the need for such metadata. Despite best efforts, Almanac still required a degree of content tagging to identify content by its educational purpose, for example as an introduction, example, case study, etc. With the specific Almanac use case in mind we wondered if it would be possible to apply ML techniques to help reduce the effort and cost involved in generating the required metadata and identified this as an ideal practical scenario within which to explore the current state of the market in commercial machine learning tools.

One of the advantages of Almanac as a case study was the availability of a data set consisting of previously tagged content slices that was generated in collaborating with CJ Fallon, our industry partner on the original Almanac project. This data set was generated by instructional design and subject matter experts and represents a ‘gold standard’ against which the results of a trained ML system can be compared.

To benefit from the ready availability of ML technology, we aimed to develop a better understanding of Machine Learning as a concept, through its application to a real-world text classification problem, using data from our collaboration with CJ Fallon. Our experiments were devised to explore the development of a text classifier that, based on relatively short items of elearning content, would be able to correctly predict which educational content tag (category) best described an item of content, based on a bespoke set of tag/categories.

1.2. Objectives

As discussed, we had two main objectives, the first was to provide a better understanding of the capabilities of available ‘off the shelf’ Machine Learning tools as it becomes an increasingly important technology. For the second objective we grounded this investigation in a practical use case, looking at the specific problem of automated content tagging, allowing for the parallel investigation of existing research and solutions to address the tagging challenge. From these objectives a number of key questions were identified:

- Can 'off the shelf' Machine Learning tools be used to tag (classify) eLearning content slices using a taxonomy of tags that do not generally appear in the content slices themselves
- What insight can we develop into the ease of use of available machine learning platforms
- How easy are these tools to use from a developer perspective?
- Howe easy would it be to integrate solutions based on these platforms into a new/existing product/service
- What other practical pros and cons are there?

In this work we asked ourselves if it is possible to automate the tagging or categorisation of eLearning content assets. As such, we focused on validating the application of Machine Learning to this real-world problem. As such our aim was not to develop a completely refined and optimized text classifier. We first adopted a somewhat naïve approach, in which we didn’t carry out a detailed analysis of the data set to identify and extract key features. but instead used the data as is before trying some basic techniques to increase the accuracy of our results.

In addition, we aim to provide a quick introduction to Machine Learning that will hopefully serve as a good starting point when considering whether ML can be an appropriate solution for the problems that you face. As such, we will introduce some of the key concepts and requirements that you need to be aware of when thinking about applying ML. This work also provided the opportunity to look at several different ML frameworks and cloud solutions, alongside their different features and accessibility to the public, and report the State of the Market.

2. A Brief Introduction to Machine Learning

Before going into the specific details of our investigation of the content tagging use case we will first discuss the main concepts of AI technology and more specifically ML by introducing the key challenges and aspects of ML algorithms.

2.1. Introduction

2.1.1. Machine Learning Concepts

Before dealing with algorithms, we will look at the problems that ML allows us to solve, and how to correctly address the issue we want our AI to provide an answer to. But let us start by talking about what is meant by Machine Learning. When we say, “Machine Learning”, what we mean is that we are training a **model**, which, given a certain ML algorithm, will **predict** with a degree of **certainty** (or confidence) an answer to our problem. For example, predict the sales of a company for the year to come. This model is different depending on the specific ML algorithm we choose to apply, but the fundamental concept remains the same, using a set of **weights** (numeric values) that the algorithm will refine through the process of **training**. There exist different ways of training a model, but they all include using many samples of **training data** that will provide examples to the ML algorithms to detect similarities and patterns in that data and hopefully correctly recognise those patterns once applying our model to **testing data or in real world application**. The data that we use to both train and test our model must contain both features that will be used as the basis for predictions and also an outcome or correct answer. If we go back to our example of predicting the sales of a company for the year to come, we are going to train our model with the sales data for the previous years. Let us say that we have access to data from the last 10 years, we can train our system on the first 7 years, and try to validate our model on the last 3. This way, we can compare our predictions with the actual sales of the company. The idea is that we want our model to **fit** our training data and **validate** our model by applying it to our testing data, to correctly **predict** any data that will be passed to our model afterwards. Once we have validated the accuracy of our system we can then apply it to our real-world application. Now that we saw the main idea of Machine Learning, let us see what problems our AI can answer using ML, in the field of data science.

2.1.2. Glossary of terms

Model: Refers to the model artefact created by the training process.
Training data: Data the learning algorithm will use to generate the model during the training process.
Testing data: Data used to test our model once the training process is over.
Weight: Numeric value part of the model, that varies throughout the training process. A model may contain a set of weights, according to the chosen Machine Learning algorithm.
Training process: It defines the process that generates the Machine Learning model. It starts by taking input data (training data), setting the target value to predict (generally a column from the training dataset) and choosing a Machine Learning algorithm that will perform the training, resulting in the model.
Model fitting: It refers to our model fitting our training data after the training process. It describes the fact that our model learnt from a specific dataset.

2.2. Issues solved with data science

Basically, there are 5 problems we know Machine Learning can solve:

2.2.1. Classification (or Multi-class Classification)

The first problem type is classification. This category can be split into 2 subdivisions, two-class (2 classes) and multi-class (more than 2 classes). Classification is the problem of determining which category our data should go into. It directly answers the following question: **is our data belonging to class A or B** (in the case of multi-class, is it A, B, C or D?)?
For instance: is this image a picture of a bird? Here we would have 2 classes, “Yes” or “No”.
Some examples of two-class classification algorithms: Two-class SVM, Two-class averaged perceptron, Two-class Bayes point machine, Two-class decision forest, Two-class logistic regression, Two-class decision tree, Two-class decision jungle, Two-class locally deep SVM, Two-class neural network.
Some examples of multi-class classification algorithm: Multiclass logistic regression, Multiclass neural network, Multiclass decision forest, Multiclass decision jungle, One-v-all multiclass.

2.2.2. Anomaly detection

The second answer data science can provide is anomaly detection and it answers the question “**is our data weird?**”. For example: is this car (defined by its component values) different than a properly functioning car?
Some examples of anomaly detection algorithm: One-class SVM, PCA-Based Anomaly Detection.

2.2.3.Regression

Another problem is regression and consists in our model to predict a certain numerical value, based on examples it has learned from. It directly answers the following questions: **How much? How many?** For instance: How much a car with those properties should cost me?
Some examples of regression algorithm: Ordinal Regression, Poisson Regression, Fast forest quantile regression, Linear Regression, Bayesian Linear Regression, Neural Network Regression, Decision Forest Regression, Decision Tree Regression.

2.2.4.Clustering

Data science also provides an answer to the following question: **How is this data organized?** Our ML algorithm is then in charge of detecting similarities in our training data and assigning them to clusters.
Some examples of clustering algorithm: K-means.

2.2.5.Reinforcement learning

The last question that data science answers is: **what should we do now?** Which action should be executed now? One of the first applications of reinforcement learning (RL) was in the scope of game theory, especially in 2-player games such as checkers, chess or go[1], (the major breakthrough of the last 5 years in RL) before being applied in many fields including robotics and industry automation. It generally involves applications where an agent interacts within an environment while trying to learn the optimal sequence of decisions.

Those problems can be solved using supervised learning (trained from labelled training data), unsupervised learning (drawing interferences from datasets consisting of input data without labelled responses) or reinforcement learning (learning by experience, by taking actions in an environment and maximise its cumulative reward).

Classification, anomaly detection and regression problems are generally solved using **supervised learning**; Clustering is solved using **unsupervised learning**. Find more information about each algorithm available on the Azure Machine Learning Studio, and when to apply them:

- https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-cheat-sheet
- https://docs.microsoft.com/en-us/azure/machine-learning/studio/basics-infographic-with-algorithm-examples

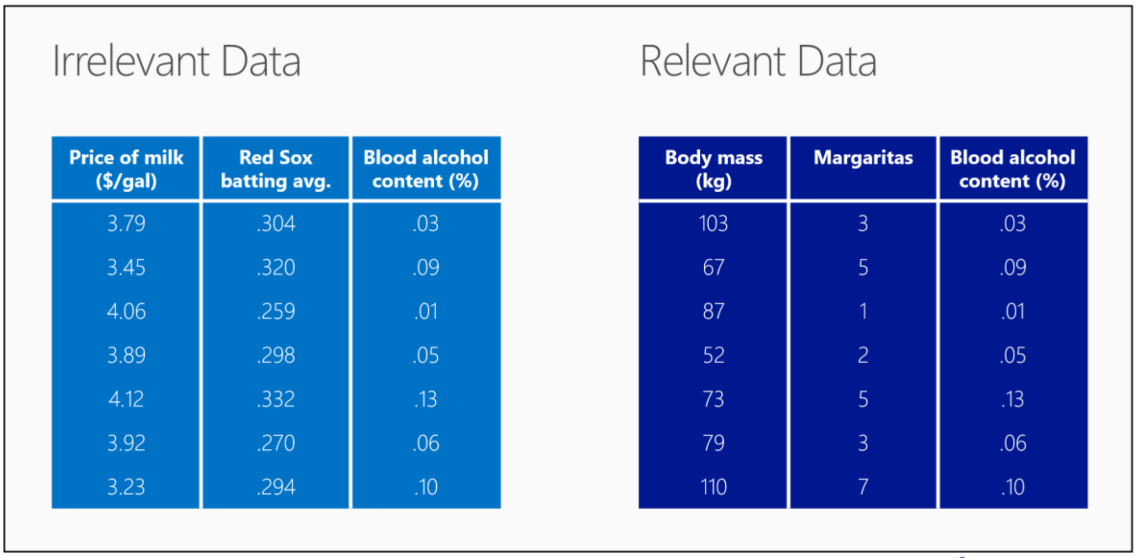
[1] https://deepmind.com/blog/alphago-zero-learning-scratch/

2.3. The importance of the data

No matter the problem we are looking at with ML, one of the key factors for applying it successfully is the quality of the data we are using. Here are the few requirements regarding data in ML.

2.3.1. Relevant

Our data needs to be relevant, which means we want data that is **related** to our problem and that has an effect on the prediction expected from our model.



Irrelevant Data

Price of milk ($/gal)	Red Sox batting avg.	Blood alcohol content (%)
3.79	.304	.03
3.45	.320	.09
4.06	.259	.01
3.89	.298	.05
4.12	.332	.13
3.92	.270	.06
3.23	.294	.10

Relevant Data

Body mass (kg)	Margaritas	Blood alcohol content (%)
103	3	.03
67	5	.09
87	1	.01
52	2	.05
73	5	.13
79	3	.06
110	7	.10

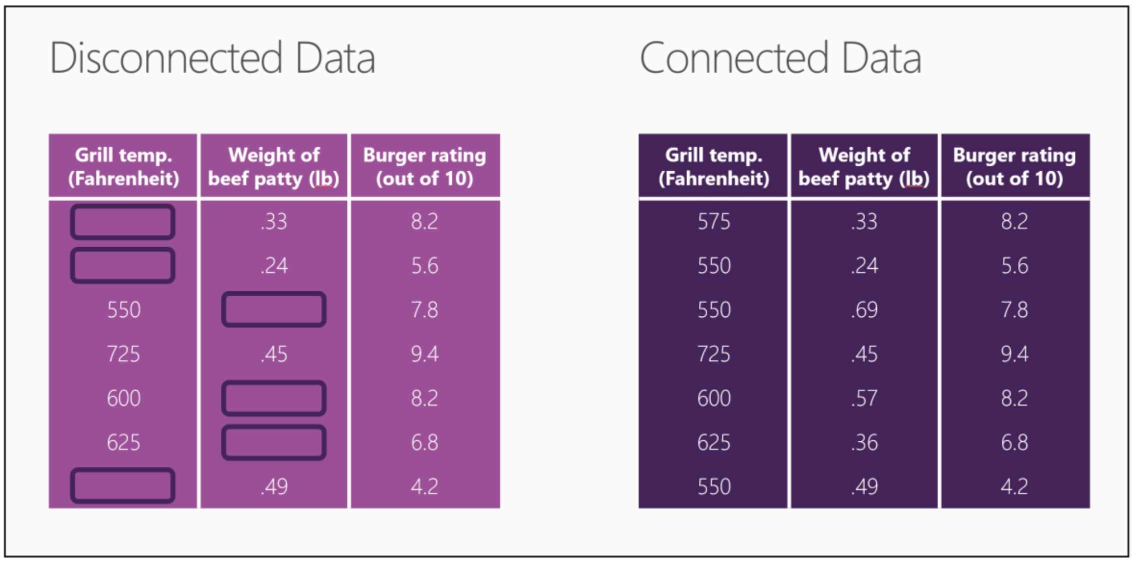
Figure 1: Comparison between irrelevant and relevant data (Microsoft data[2])

If we look at the example in Figure 1 (from Microsoft documentation), we can see the difference between irrelevant data (left side) and relevant data (right side). In this example we want our model to predict blood alcohol levels. The body mass and the number of Margaritas that a person drank are more relevant than the price of milk in the nearest convenience store and the Red Sox baseball team batting average in their last game when it comes to predicting blood alcohol content. This might seem intuitive but sometimes it can take careful consideration to determine which information is relevant to the problem in hand.

[2] https://docs.microsoft.com/en-us/azure/machine-learning/studio/data-science-for-beginners-is-your-data-ready-for-data-science

2.3.2. Connected

It has to be connected, which means having data as **complete** as possible, to avoid any bias that would occur in cases where information is missing for some items in our samples.



Disconnected Data

Grill temp. (Fahrenheit)	Weight of beef patty (lb)	Burger rating (out of 10)
	.33	8.2
	.24	5.6
550		7.8
725	.45	9.4
600		8.2
625		6.8
	.49	4.2

Connected Data

Grill temp. (Fahrenheit)	Weight of beef patty (lb)	Burger rating (out of 10)
575	.33	8.2
550	.24	5.6
550	.69	7.8
725	.45	9.4
600	.57	8.2
625	.36	6.8
550	.49	4.2

Figure 2: Comparison between disconnected and connected data (Microsoft data[3])

In the example shown in Figure 2, data is relevant, however, on the left-hand side, the many missing values make that data disconnected, as opposed to the data shown on the right.

2.3.3. Accurate

The accuracy of data is determined by 2 main factors, its value must be **correct**, and its representation must be **consistent** (same formatting for all entries of the dataset). For example, if we take the above table of hamburger data, and consider how it might look both when represented accurately and inaccurately. Table 1 provides an example of accurate data in which everything is consistent and looks how we would expect it while in Table 2 the data is inaccurate. We can see from the values highlighted in red that some of the Grill Temperatures are plainly incorrect, we must have introduced some errors as we exported the data from the original data source into our table. A subtler issue occurs with the weights reported. They should all be recorded in pounds (lb) but some of the weights are missing a decimal place. This has the potential to adversely affect any training of a ML model based on this data and may change the overall predictions of our model.

[3] https://docs.microsoft.com/en-us/azure/machine-learning/studio/data-science-for-beginners-is-your-data-ready-for-data-science

Table 1: Accurate representation of hamburger data

Grill temp. (Fahrenheit)	Weight of beef patty (lb)	Burger rating (out of 10)
575	0.33	8.2
550	0.24	5.6
550	0.69	7.8
725	0.45	9.4
600	0.57	8.2
625	0.36	6.8
550	0.49	4.2

Table 2: Inaccurate representation of hamburger data

Grill temp. (Fahrenheit)	Weight of beef patty (lb)	Burger rating (out of 10)
900	**33**	8.2
5500	24	5.6
550	0.69	7.8
680	45	9.4
560	0.57	8.2
6250	0.36	6.8
275	0.49	4.2

2.3.4.Enough to work with

Of course, the quantity of data we provide also matters, and the higher the **quantity of data** the better coverage our model will have. By ‘coverage’ we mean that that we want our model to have a complete overview and representation of the data. Note that having more training data is *always* a good thing, as long as it meets all the above data requirements. On a more technical note, it is also a way of counteracting over-fitting, which represents the production of an analysis that corresponds too closely or exactly to a particular set of data, and that will result in a less efficient model when predicting data. Over-fitting happens when data is complex, but there is not enough of it, so the model fits our training data too well (over-fits), and therefore becomes inefficient when applied to other data[4].

As said previously, ML algorithms learn to predict answers to specific questions from training data. Models are generated by analysing the **features** of that data (if we take cars as an example, the make, the cylinder volume, the maximum speed, etc… can be seen as features) in order to predict the **target** value (for example, the price of the car, in case of a regression problem, or a certain category

[4] http://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

in a classification one). Therefore, it is recommended to favour or select the most relevant features that helps better our problem.

2.4. Defining the problem

Another key element of ML is to address concisely the problem you want an answer to by:

- Formulating a very specific question to get the most helpful result.
- Knowing what kind of result (target data) you are hoping to get: numerical value, category, action, etc.
- Reframing problems (classification problem into regression problem and vice versa).

3. Commercially Available Content Tagging Services

We have identified the problem of automating the tagging or categorisation of eLearning content as a problem that we believe ML can help to address but surely, we're not the first people to think of this. Before trying to solve this problem we first asked ourselves whether any services existed that already provided a solution to this problem or provided functionality that could readily provide us with the solution we were looking for. Rather than completing and exhaustive state of the art review including the latest in academic and commercial research we constrained our focus on commercially available solutions that were readily available and that represented a practical solution.

When considering the tagging of content resources such as those found in eLearning there are generally two broad approaches that can be taken depending on the specific problem that you want to solve. The first is Entity Extraction, which uses ML and Natural Language Processing (NLP) techniques to extract information or 'entities' from the text relating to people, places, objects, concepts/keywords, etc. that are mentioned. This can be a powerful tool in automating the generation of metadata to describe a resource or document but it only works when the information that you want to extract is contained within the document. The second approach is Text Classification, which is intended to categorise an item of text into one or more predefined categories (called classes in the scope of Machine Learning). This approach is appropriate for scenarios where the documents don't necessarily refer explicitly to the classes, for example if a newspaper article should be in the news, sports, business or entertainment section. It is the latter 'text classification' problem that is the focus of our interest as it aligns with the problem that we're trying to solve.

Several commercial Text Analysis/Classification APIs are currently available that provide a range of basic functionalities. They can extract entities and keywords from a piece of text, analyse the positive or negative sentiment of a text, detect similarities between two texts or classify text according to a set of predefined classes.

As part of this survey of available commercial services we identified a number of the larger offerings, some of which come from the familiar multinationals such as Google and Microsoft while others are more focussed ML services.

- Dandelion API[5]
- Aylien Text Analysis API[6]
- Microsoft Text Analysis API[7]
- Google Cloud Natural Language API[8]
- TextRazor[9]
- ParallelDots[10]

The majority of the APIs that we reviewed offer a free subscription plan for users to experiment with before requiring a paid plan in order to scale up to a higher number of daily transactions (API calls to the ML service) when moving from development to production. Here is a comparative table of the different pricing plans:

All APIs except the Microsoft Text Analytics API provide a Text Classification feature. However, these classification APIs are restricted to only making tagging suggestions from a predefined taxonomy (such as IAB QAG and IPTC Subject Code). The ML engines have been trained in advance based on the companies' own training data. This implies that those APIs are using a model that has been trained on previous data **before** you can test it with your own custom data. The implication of this is that the API only has a relatively narrow field of use. For example, Aylien provide a Text Classification API that has been trained using the IPTC international Subject News Codes. This can categorise text according to 500 different news article categories (e.g. sport, soccer). However, unless this is the functionality that our application requires, this API is of limited use. Classifiers of this type are obviously not suitable for every custom application. Where we want to classify our own text-based documents using a taxonomy that is specific to our use case we need to fall back to the more generic ML services and train our own system.

Another possible option to consider would be to select one of the semantic labelling applications, which belongs to the clustering category (unsupervised learning). In that case, the solution would draw similarities (in our case, based on the semantics of the classes we would provide) in order to predict the output label. However, this method would not achieve high accuracy, due to the fact that semantics would not perform well with contextual concepts such as Definition, Exercise, Example or Keywords. Put very simply, the text based data that we want to categories (the eLearning content) does not generally or reliably contain words or phrases that can be semantically linked to the

[5] https://dandelion.eu/
[6] https://aylien.com/text-api/
[7] https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/
[8] https://cloud.google.com/natural-language/
[9] https://www.textrazor.com/
[10] https://www.paralleldots.com/text-analysis-apis

categories that we want to tag the content with. Simplistically, we wouldn't generally expect an item of learning content designed to communicate practical knowledge about a topic to explicitly state this to the learner.

For those reasons, the only way to address the specific problem that we want to solve with our data would be to train our own model, using tagged content as training data (supervised learning).

Table 3: Comparison of existing Text Analysis APIs[11]

API Name	Transaction per day (free)	Transaction per month (free)	Price per 1000 requests	Features
Dandelion	1K	30K	0.375$/1000 hits (2 million requests per month) ➔ 750$/month[12]	Entity extraction, Text and content Classification, Sentiment Analysis, Keywords extraction, Semantic similarity, Article extraction
Aylien	1K	30K	0.27$/1000 hits (2.4 million requests per month) ➔ 650$/month[13]	Sentiment analysis, Text Classification, Entity Extraction, Content Processing (article extraction, suggestion and generation)
Microsoft Text Analysis	<200?	5K	2$/1000 hits (500K requests per month) ➔ 1000$/month[14] 1$/1000 hits (2.5 million requests per month) ➔ 2500$/month	Sentiment analysis, Key phrase extraction, Language detection
Google Cloud NLP	<200? 1K?	5K 30K	<=1$/1000 hits depending of the API call (1 million requests per month)[15] 2$/1000 hits (250K requests per month)	Entity, Sentiment, Syntax and Entity sentiment analysis Content Classification
TextRazor	500	15K?	0.4$/1000 hits (1.5 million requests per month)? ➔ 600$/month[16]	Entity extraction, Key phrase extraction, Automatic Topic tagging and classification
ParallelDots	1K	10K	1$/1000 hits (up to 1 million requests per month)[17]	Sentiment Analysis, Entity extraction, Text Classification, Keyword Extractor, Emotion Analysis, Semantic Analysis

[11] Pricing correct as of Feb 2018
[12] https://dandelion.eu/profile/plans-and-pricing/
[13] https://developer.aylien.com/plans
[14] https://azure.microsoft.com/en-us/pricing/details/cognitive-services/text-analytics/
[15] https://cloud.google.com/natural-language/pricing
[16] https://www.textrazor.com/plans
[17] https://www.paralleldots.com/pricing

4. Approach

We will approach our two main objectives in reverse order, looking first at whether or not we can apply ML techniques to support the tagging of educational content. As mentioned previously, the aim here is not to develop a production ready solution but to technically validate the concept as a precursor to any future refinements of the solution. To achieve this, we will select one of the available ML services as our experimental platform and iteratively carry out a number of experiments allowing us to compare the effectiveness of different ML techniques and approaches to processing the available data. We have selected the Microsoft Azure Machine Learning Studio, a feature of the Microsoft Azure cloud-computing platform, for this purpose as it provides a rich and intuitive user experience that is ideal for the purposes of rapid experimentation.

The basis for this experimentation is a data set provided by CJ Fallon that was derived from work on the Learnovate Centre's previous Almanac project, which they also actively contributed to. The data set consists of short items of learning content for Junior Cert school students in Ireland that have previously been categorised based on a bespoke taxonomy of pedagogical tags. The original source for these content items were text books that were broken up into small 'slices' for the purposes of Almanac. The data set actually consists of four separate collections of content, with a different taxonomy of pedagogical tags used to categorise the content in each. For the Almanac project the categorisation of content items was carried out manually by a single subject matter expert. Four different subject areas are covered namely Geography, History, Business Studies and Science. Each item of content is in the form of a HTML document and as such has some basic formatting. For each item of content, we have the following information:

- Title
- Text Content
- Pedagogical Tag (category)
- Unit of Learning (e.g. chapter in a text book)

Table 4 below describes the data sets available:

Table 4: Available datasets for our experiments

Subject	Number of content assets	Number of pedagogical tags (classes
Geography	242	11
History	960	9
Business	469	10

With the luxury of having four different data sets we can run a number of experiments to investigate the effectiveness of the approach across content with different characteristics. This should help to ensure that our results are more generally applicable and not overly specialised to the specific characteristics of a specific style of content.

A standard approach in training machine learning systems is to take the available data set and split it into two, training data and test data. The training data, as its name suggests, is used to train the system to hopefully categorise the content while the test data is used to evaluate how well the system performs at this task. The test data is then used to evaluate the accuracy of the predictions made by our newly trained ML model.

Several experiments will be carried out to investigate a range of different characteristics including how much data it takes to train the system before it becomes sufficiently accurate to be useful and which features (title, unit of learning, etc.) have an impact on the accuracy of the predictions.
Our first experiments will be run based on a naïve approach, treating the text content as a single feature without any analysis or refinement. This would normally not be a sensible first step but we are interested in how treating the entire text of a content slice as a single feature will affect the outcome. This initial round of experiments consisted of the following:

- **With title and topic**, our model will be based on taking the title and the topic of a content item into account in order to predict its result (2 features: title and topic name).
- **With text and topic**, this time the article text and topic will be taken into account (2 features: text and topic name).
- **With text, title and topic**, here the 3 parameters will be associated (3 features: text, title and topic name).

Our assumption before running these experiments is that the accuracy of our predictions should increase when including the content text and mixing both title and text.

After running the initial, naïve experiments, our second round of experiments will look at how pre-processing the text, hashing features (extracting different feature from the text itself) and selecting the most promising features (some features being more useful and distinctive than others) impacts on the accuracy. Our assumption is that, if done properly, the results should improve, as we assume that the ML algorithms will make better use of additional, more refined features rather than just a single plain text feature.

The final step in our experimentation will be to optimise how the best performing ML algorithm performs and to validate the performance across the different subject areas that we have data sets for.

All of the experiments will consist of 5 major steps:

- Data preparation (import the dataset, clean the data and pre-processing in case our data is text).
- Feature selection (in case of text, title and topic name)
- Determine training and testing data (in our case, split the sample into 2 categories)
- Select one Machine Learning (ML) algorithm (Neural Networks, Logistic Regression, Decision Forest, Decision Jungle, One-vs-All, …).
- Scoring the model based on the predetermined testing data and evaluation of the model.

Following the experimentation with Microsoft Azure Machine Learning Studio we will hopefully have demonstrated that ML can produce meaningful results when categorising the type of content that we are interested in. The next step will be to look at other available Machine Learning frameworks so that we can investigate the practical implications of using these tools. We will be looking here at 2 additional frameworks for running our experiments. Our approach will be to take the best configuration that we developed through our initial experimentation and try to reproduce those experiments using the Python programming language and some of the many Machine Learning and data science libraries that are for available (including TensorFlow[18], numpy[19] and pandas[20]). Finally, we will look at Google Cloud and their own machine learning tools[21].

[18] https://www.tensorflow.org/
[19] http://www.numpy.org/
[20] https://pandas.pydata.org/
[21] Ideally, we would have extended this review to other platforms such as Amazon Web Service but this was not viable within the scope of this project

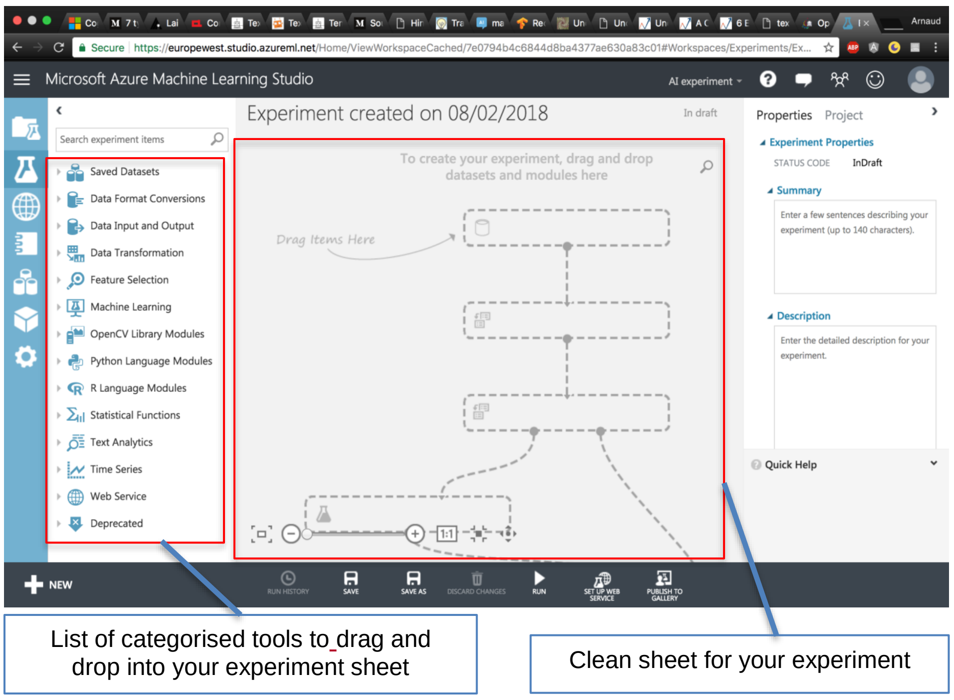
5. Introduction to Azure Machine Learning Studio

The Azure Machine Learning Studio is a cloud based Graphical User Interface (GUI) environment allowing developers and data scientists to build, test and deploy predictive analytics solution on their own datasets. It is a browser-based, visual, drag-and-drop tool giving the opportunity to build simple ML models without the need for coding. At the same time, it provides the flexibility to be able to go beyond the tools it provides by default in order to extend its functionality when necessary.

You can import your data, make transformations on it, train your model and evaluate its performance by dragging elements into a clean sheet representing the flow of your experiment. This represents a great platform for running experiments on your data as part of any initial design and development work as it greatly simplifies the task of changing the configuration of your ML solution and comparing the results obtained. In fact, it includes specific functionality for comparing the outputs of different approaches across the same data set. Azure Machine Learning Studio also allows you to publish your predictive experiment as a web service. Once you are satisfied with the result, the trained ML models can be easily integrated into new and existing solutions. This, along with the ability to access large datasets using cloud based data storage, illustrate some of the advantages of cloud based solutions.

When starting your own experiment, a menu of tools (dataset, Machine Learning algorithm, training method, data-manipulation operations, etc…) is provided, broken down into categories. You can directly drag and drop these tools into your experiment as illustrated in Figure 3 below. A more complete description of Azure Machine Learning Studio can be found in Appendix A.

Figure 3: Experiment view on the Machine Learning Studio



6. Content Tagging Experiments

Azure Machine Learning Studio is quite flexible, allowing users to run a whole range of experiments easily. With some basic knowledge, and with the help of Microsoft's online technical documentation, it is possible to run various experiments, compare their results and have a better understanding of what happens when training the ML models. However, optimizing the models requires a better knowledge of machine learning fundamentals to configure the different parameters of the modules involved in each experiment. Azure Machine Learning studio also provides many ways of manipulating data, such as filtering methods or data cleaning operations.

For our experiments, we took a percentage of the whole dataset as training data. Table 5 provides details of the specific number of content assets that equate to a given percentage of the. The numbers reported here are for the Business Studies content as this will be the basis for our initial experiments.

Table 5: Percentage breakdown of assets from Business Studies collection

Percentage	Number of samples (assets)
5%	24
10%	47
25%	118
50%	235
75%	352
85%	399
100%	469

6.1. Results from initial experiments

We ran 3 batches of experiments as part of our naïve approach using the following features:

1. With title and topic name
2. With text and topic name
3. With title, text and topic name

For each round of experiments, we used the Business Studies content collection and applied 4 different ML algorithms on the data: Neural Network, Logistic Regression, Decision Forest and Decision Jungle. We applied the default parameters recommended by Azure Machine Learning Studio for each algorithm. Each experiment was carried out with three different sizes of training data at 50%, 75% and 85% of the overall dataset respectively. Table 6 summarises the results from the initial 'naïve' set of experiments in which no analysis or feature extraction was carried out on the textual content from our dataset. The percentages shown represent the *overall accuracy* for each algorithm on the 3 different variations (title & topic, text & topic, title & text & topic).

The overall accuracy is a way of measuring the performance of our model, and represents the number of correctly predicted items out of the overall number of items (it is also referred to as the precision). A finer level of detail for each experiment can be found in **Error! Reference source not found.** where the results of each experiment are broken down per category and the Average Accuracy is reported as well as the Overall Accuracy.

Table 6: Comparative table of the overall accuracy from initial experiments on Business Studies content collection

	Title & Topic				Text & Topic				Title, Text & Topic			
Training Data Percentage	Neural Network (1000 iterations)	Logistic Regression	Decision Forest	Decision Jungle	Neural Network (1000 iterations)	Logistic Regression	Decision Forest	Decision Jungle	Neural Network (1000 iterations)	Logistic Regression	Decision Forest	Decision Jungle
50%	41%	42%	41%	39%	35%	33%	34%	33%	42%	37%	39%	37%
75%	48%	42%	44%	43%	35%	35%	41%	33%	45%	39%	41%	39%
85%	7%	4%	7%	3%	37%	37%	41%	38%	51%	41%	37%	38%

We can make 4 observations based on those results:

- The predictive models worked best when using the title and topic only at 50 and 75%, which could seem surprising when compared with the model using all three features, title, text and topic combined. This may be due to the quality of titles, and the fact that many of them are very similar or even containing the name of the tag itself in it (E.g.: "Keywords", "Example", "Exercise"). That being said, maybe our results would be different with less explicit data.

- The model seems to 'over-fit' with title and topic only at 85% of training data, as illustrated by the significant drop in accuracy from as much as 47% down to only 7%. Our assumption here is that it could be due to the length of the title, and the fact that many of them were identical, as opposed to full chunks of text. However, each time we include text, results get higher when we increase our training data percentage, the best result being with 85% of training data.

- None of the models perform especially well with the highest accuracy being 51%. Although not a great result this is still significantly better than a random guess, which would only be accurate 18% of the time. The poor performance is due to the fact that the data is broad text, therefore too scattered for the model to detect patterns in it and difficult for our Machine Learning algorithm to interpret meaningful information without assistance. Remember, we made no attempt to extract meaningful information from the text and simply treated it as a single feature.

- The Neural Network is so far, the algorithm performing best.

The first round of experiments showed us that training using text only is less effective than training using the content title alone irrespective of the algorithm used. It also showed that training those systems with both text and title is actually less accurate than using only the title (until it seems to over-fit with 85% of training data). These results need to be treated with caution, and several factors should be considered:

- **Quality of the title** If our experiments including the content title performed better than the ones without, it is mostly because of quality of those titles, and the fact that many of them are self-explanatory (Keywords, Exercise). In other circumstances, our system would probably be less accurate, or more random.

- **Length, accuracy of the data (text)** Adding the text into the evaluation of the content to be tagged is more than relevant, but the text itself needs to be parsed, cleaned and pre-processed in order to get better results with it. The cleaner and more accurate the data is, the better our results will be.

- **Raw data** Not only is our data not of the best quality at the moment, but it is also only raw text. One solution would be to stem our text. This process involves converting each word of our text into the root of the given word. For example: “the cars are different colours” would turn into “the car be differ colour”. The goal of stemming is to remove derivational affixes, to simplify our text and reduce the range of words used in our text, to simplify it. However, this process will not be enough to increase significantly our results, as it won’t draw obvious enough patterns. We might need to manipulate the raw data by applying some text analysis algorithms (N-grams, entity extraction…) in order to get better results. The aim of this would be to take the single feature of our raw text and turn it into many different features that the ML algorithms can interpret.

6.2. Improving our results through feature hashing and selection

Based on our initial experiments we can see that running our experiments using 75% training data gives us the best average score, offering great comparative scores and avoiding over-fitting. Therefore, we will only compare the scores of experiments run with a data split of 75% of training data and 25% (117 samples in the case of our Business Studies collection) of testing data for future experiments. Note that during our data split, we distribute the samples in a way that each class is apportioned equally among both training and testing data.

In this round of experiments, we will attempt to build on what we have learned so far to improve the accuracy of our predictions. To achieve this, we will attempt to carry out some basic analysis of the text that makes up our data set. Having said that we will still restrict ourselves to using basic tools that are available to use within Machine Learning Studio. We will not carry out a more complete analysis of the dataset as we would if were moving beyond an initial proof of concept.

The first step in this analysis was to generate a hash for each of the features (title and text). The advantage of using feature hashing (with n-grams) is that you can represent text documents of variable-length as numeric feature vectors of equal-length, and achieve dimensionality reduction. In contrast, when we used text columns for training as is, they were being treated as categorical feature columns, with many, many distinct values[22].

The second step was to select the most distinctive features from the ones generated by the feature hashing. Here, the goal was to make a distinction between relevant and irrelevant features, according to a feature scoring method (we used the Chi squared method).

Here is the experiment we tried for each algorithm (we took the example using Neural Networks and Logistic Regression for the following screenshot):

- Import the data (CSV file)
- Convert the data into a usable dataset
- Clean the data by removing lines of the dataset containing missing information
- Pre-process the text (article content)
- Hash the features generated using n-grams (title of the article, pre-processed text) (1)
- Select filtered features according to a scoring method, allowing ML algorithms to draw patterns from (2)
- Select those columns (features previously featured) and split the data (75% of training data, 25% of testing data) for each ML algorithm

[22] https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/feature-hashing

- Train the model (according to the ML algorithm) with the training data
- Score the model on testing data
- Evaluate the model, and see how it performed on the testing data

(1) To generate more features from our text, we used the n-grams technique, which turned our text into many pieces of text, each piece of text becoming a new feature. The goal here is to help our model using the text for its prediction. We saw previously that using the text as is was ineffective. Therefore, we are trying here to decompose our text (into small sequences of words), so that our model could learn to detect which sequences of this text are the most distinctive in it. The feature hashing process allows the vectorization (transform the text feature into a numerical feature) of the text column. Note that there are several parameters for the feature hashing process with the Machine Learning Studio. In our case, we started at 15 as the hashing bitsize, and 2 N-grams.

(2) This step enables the filtering of the most relevant features to take into consideration for the ML algorithm later on. In our first experiment, we used the Chi Squared featuring scoring method.

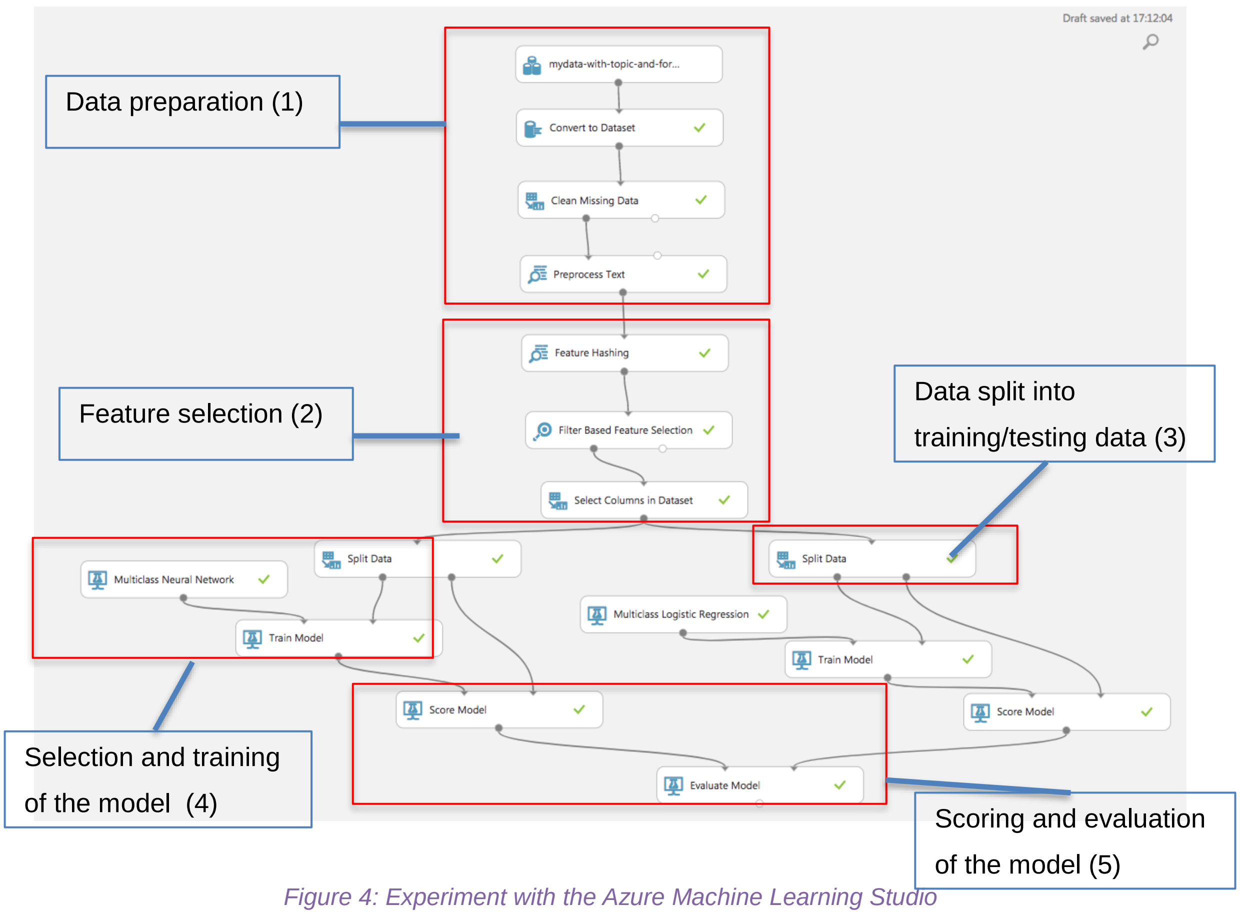

Draft saved at 17:12:04
Data preparation (1)
mydata-with-topic-and-for...
Convert to Dataset
Clean Missing Data
Preprocess Text
Feature Hashing
Filter Based Feature Selection
Select Columns in Dataset
Feature selection (2)
Data split into training/testing data (3)
Split Data
Multiclass Neural Network
Train Model
Split Data
Multiclass Logistic Regression
Train Model
Score Model
Score Model
Evaluate Model
Selection and training of the model (4)
Scoring and evaluation of the model (5)

Figure 4: Experiment with the Azure Machine Learning Studio

Figure 4 provides an illustration of the experiment workflow including the feature hashing, etc. Note that we are scoring and evaluating two models simultaneously here (Neural Network on the left and Logistic Regression on the right). Therefore, except for the algorithm choice (Multiclass Neural Network and Multiclass Logistic Regression), the right and left sides are identical. To get the results reported below, we ran the same experiment, with the 4 different algorithms.

Table 7 summarizes the results from these experiments, showing the overall accuracy of each algorithm, after feature hashing and filtering, when applied to title & topic, text & topic and title, text & topic. This is followed by our observations based on those results. When evaluating each of the different ML algorithms we used a 75% training data - 25% testing data distribution. From the initial round of experiments, it was observed that this split generally returned the best results. The table also compares the results obtained against two baselines, a random guess and the so called zero rule. The random guess gives us the percentage of times that we would correctly guess the category if we just did it randomly while for the zero rule we always guess the category that contains the most items from our dataset. In this case the 'Exercise' category makes up 31% of the Business Studies collection so we would get an accuracy of 31% just by assuming this was the answer every time. The detail of each experiment is reported below the table.

Table 7: Recapitulative table for our experiments after Feature hashing and filtering (Business Studies data set)

ML Algorithm (75% training data)	Title & Topic	Text & Topic	Title, Text & Topic	Baseline	
				Random Guess	Zero Rule
Neural Network	64%	62%	78%	18%	31%
Logistic Regression	70%	56%	74%		
Decision Forest	70%	58%	69%		
Decision Jungle	62%	47%	64%		

The observations we can make here are the following:

- Unlike previously with the naïve approach, our predictions are **higher when applied to title and text combined**. This is due to the fact that we prioritized the features that differentiate the most our data via feature filtering. So, although the results for 'title & topic' and 'text & topic' are similar (especially for the Neural Network) we still get a beneficial cumulative effect when we use both title and text at the same time. This indicates that there are cases where the features of the text are much better at predicting the correct category.

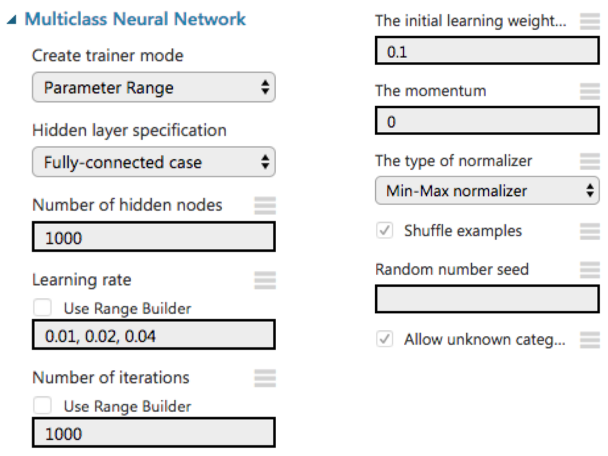
- **Neural Network is still the best-performing algorithm** with a 78% overall accuracy with Title, Text and Topic.

- Even though our overall accuracies increased compared to the approach in our first experiment, some classes (tags) were never predicted (such as Case Study, Rationale or Summary, see Appendix C). This phenomenon seems to be due to 2 main factors: the **uneven distribution of training data** compared to other classes (1) and the **lack of distinctive features** for those classes (2).
  The uneven distribution of training data can be an obvious reason for the poor prediction success rate on those classes, when we consider that there were only 4 samples belonging to the "Summary" class in our training data, as opposed to the 108 training samples for the "Exercise" class. But this is not the only reason explaining our results. Indeed, if we compare the "Keyword" and "Rationale" classes, the amount of training samples was very similar for both classes (15 for "Rationale" against 16 for "Keywords"). However, as you will see in the detailed experiment results in Appendix C (combining text and title), our accuracy for those 2 classes were drastically different (80% of our testing data were correctly predicted for the "Keyword" class when not even 1 piece of content was correctly predicted for the "Rationale" class). This phenomenon is due to the fact that pieces of text belonging to the "Keywords" class are very distinctive (generally speaking, the title of "Keywords" content contains the word "keyword" in it), therefore, our model is able to detect this similarity in those contents. The fact that "Rationale" content is less distinctive makes those pieces of content much harder for our model to predict.

- By carrying out some basic pre-processing in the form of feature hashing and filtering we have been able to improve the accuracy (with 75% training data) from 51% to 78%.

6.3. Tuning model parameters

Our previous experiments highlighted our model's improvements through feature hashing and filtering. But despite the good progress, our models are unlikely to be optimal as we haven't tuned each parameter to find the best combination for each algorithm. Going through each parameter seems a heavy task to carry out but thankfully Azure Machine Learning Studio provides an easy way of achieving that for us called "Tune Model Hyperparameters". It allows a range of values to be set instead of a single value for each parameter, the result is that this tuning module helps to identify which set of parameters offered the best results (according to a single metric previously selected: precision, accuracy, recall, etc.). Note that it is possible to use it with each ML algorithm.


Multiclass Neural Network

Create trainer mode: Parameter Range

Hidden layer specification: Fully-connected case

Number of hidden nodes: 1000

Learning rate: Use Range Builder ☐ 0.01, 0.02, 0.04

Number of iterations: Use Range Builder ☐ 1000

The initial learning weight...: 0.1

The momentum: 0

The type of normalizer: Min-Max normalizer

✓ Shuffle examples

Random number seed:

✓ Allow unknown categ...

Figure 5: Example of range of parameters to tune models on the Machine Learning Studio

This kind of experiment takes more time to complete, as several combinations of values have to be tested in order to produce the best outcome for the model. This is still a very convenient feature as it automates the execution of a whole set of experiments that you would otherwise have to configure manually and record the results for later analysis.

Here are the optimal combinations for each algorithm tested with 70~75% training data. Note that we included the One-vs-All multiclass ML algorithm (using Decision tree), which turned out to have equivalent results to the Neural Networks'. The One-vs-All strategy involves training a single classifier per class (which, in the case of the Business Studies collection, means fitting 10 classifiers). This strategy requires the base classifiers to produce a real-valued confidence score for its decision, rather than just a class label; discrete class labels alone can lead to ambiguities, where multiple classes are predicted for a single sample. Therefore, each classifier gives a score between 0 and 1; the highest value becomes the predicted label. The higher this confidence value is, the more confident our model is in its prediction, and so we should be able to trust this prediction is correct.

For example, in the case we have 3 classes (“Example”, “Exercise” and “Keyword”) and we try to classify a piece of content, our 3 classifiers will give a prediction score. Let us take imaginary values for our example: the “Example” classifier returns a value of 0.17, the “Exercise” classifier gives us 0.59 and the “Keyword” classifier returns 0.24. The predicted label for our piece of content then becomes “Exercise”. This real-valued confidence score also gives us an idea of how certain a particular prediction is (here, 59%).

6.4. Improving the efficiency of the prediction

We have now tuned our algorithms so that they perform in a more optimal way with the data in our dataset. The following experiments will be devoted to increasing the efficiency of our predictions without modifying any parameter of our model.

To do so, we will be focusing our experiments on data improvement, feature hashing and additional text analytics. In these experiments, we will apply Neural Networks and One-vs-All (coupled with a Two-class Decision Tree classifier) models, as they proved themselves to be the most efficient, given the amount and quality of our data, in their predictions, both performing at 78%.

6.4.1. Optimising n-gram length

The first optimisation we investigated was the length of the n-grams generated from the text. By default, the system was generating bi-grams and this is what we used in our second round of experiments. The length of the n-grams generated was iteratively increased and used to train and evaluate the models. The results of this testing are presented in Table 8. As shown, the optimal length of n-gram for the dataset is a 9 words sequence.

Table 8: Comparison of Neural Networks and One-vs-All accuracy when modifying n-grams (Business Studies)

N-gram length	Overall Accuracy (Neural Network)	Average Accuracy (Neural Network)	Overall Accuracy (One-vs-All)	Average Accuracy (One-vs-All)
1	72%	94%	74%	95%
2	78%	96%	78%	96%
3	78%	96%	80%	96%
4	78%	96%	81%	96%
5	80%	96%	76%	95%
6	84%	97%	74%	95%
7	81%	96%	74%	95%
8	85%	97%	75%	95%
9	85%	97%	76%	95%
10	84%	97%	76%	95%

🟧	Initial parameters
🟩	Best result

6.4.2.Test on different content collections (Science and History collections)

For a matter of time, we will experiment with Neural Network only, as training One-vs-All models are very time-consuming, and since the Neural Network approach obtained the best set of results so far. Once again, we will make the n-grams vary, to compare the range of prediction percentages with the previous one. Note that the Science collection is significantly bigger than the Business Studies Collection (725 samples for Science against 469 samples for Business Studies). Also, the number of tags (or classes for our classifier) is larger (10 for Business Studies against 13 for Science), which might increase the difficulty for our classifier to correctly predict some classes.

The results for the Science collection are shown in Table 9. Using the same parameters and architecture for our Neural Networks on both collections, we can see that the n-grams combination that works best in average for both use cases is the 6-grams feature hashing, obtaining 84% of accuracy with the Business Studies, and almost 70% for the Science collection.

The accuracy of the prediction is significantly inferior for the Science collection. This can be explained in part by the higher number of classes for the Science collection, therefore predicting one class out of 13 is harder than predicting one class out of 9 or 10 classes.

Table 9: Accuracy of Neural Networks when modifying n-grams (with the Science collection)

N-gram length	Overall Accuracy (Neural Network)	Average Accuracy (Neural Network)
1	63%	94%
2	68%	95%
3	68%	95%
4	64%	95%
5	63%	94%
6	69%	95%
7	67%	95%
8	63%	94%
9	65%	95%
10	65%	95%

🟩 Best result

We also applied the same experiment to the History collection, which is once again larger than the 2 we have previously worked with (over 960 content samples). Note that the number of classes (tags) in the History collection is similar to the Business Studies collection (9 classes for History collection compared to the 10 for Business Studies).

We can see that our results are similar to those that the we obtained originally with the Business Studies collection. Therefore, we could be tempted to establish a correlation between the similar amounts of classes for both collections, reducing the percentage of errors.

Table 10: Accuracy of Neural Networks when modifying n-grams (with the History collection)

N-gram length	Overall Accuracy (Neural Network)	Average Accuracy (Neural Network)
1	66%	92%
2	74%	94%
3	72%	94%
4	74%	94%
5	77%	95%
6	78%	95%
7	78%	95%
8	77%	95%
9	75%	95%
10	76%	95%

🟩	Best result

6.4.3. Comparison of training data split with Neural Networks (6-grams)

In this experiment, we used the same Neural Network experiment, varying the data split, in order to compare the efficiency of the model with regard of the amount of training data required. Based on our previous batch of experiments, we decided to apply 6 grams for our feature hashing process, as it showed the best results with the Science and History collections, and the second best with the Business Studies collection.

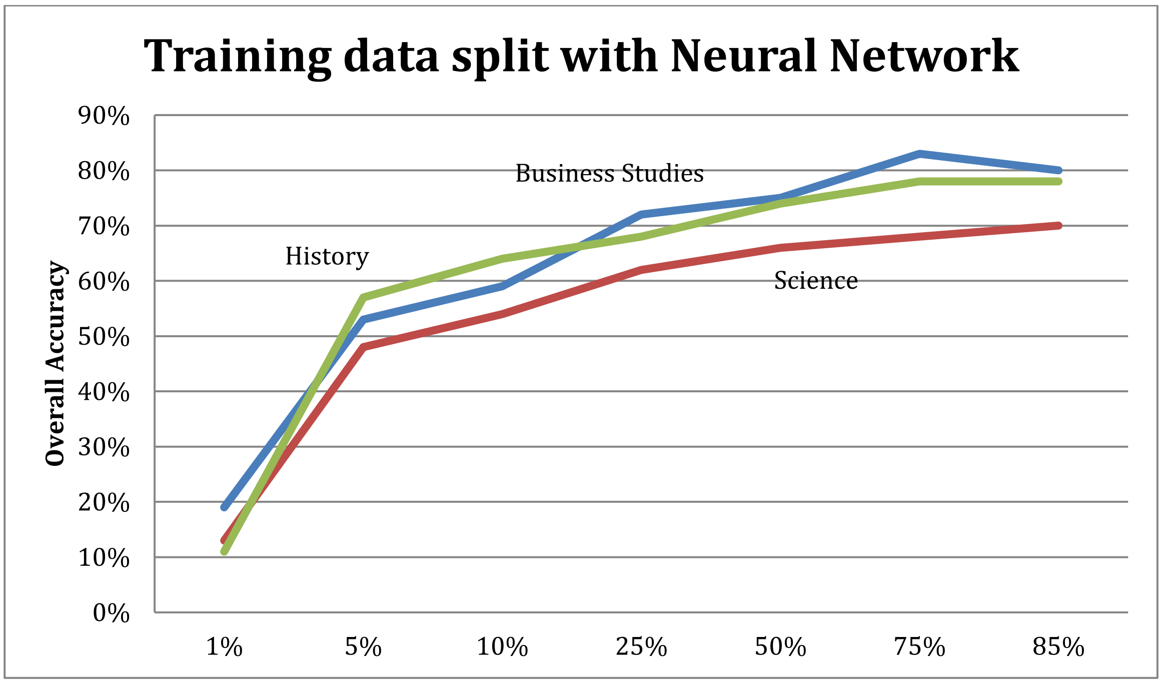


Figure 6 Effect of training data size used on overall accuracy of model

Figure 6 illustrates the effect that the amount of the training data used to create our model has on its overall accuracy. We can see similar trends when comparing all three subjects. Our classifier approaches 50% of correct predictions using only 5%[23] of training data and keeps increasing until ~75%[24] of training data. When going above 85%[25] of training data, the testing data becomes insufficient and therefore may not reflect accurately the proper efficiency of our model. Our results become therefore too biased for establishing any conclusion. This is an important finding as it indicates that there might be a practical application of this approach in the real world. With relatively little training data we begin to approach levels of accuracy that would be useful, perhaps not in a fully automated use case but they could be highly valuable in a semi-automatic approach where the ML is providing assistance to a human. For example, 25% of the data corresponds to 118 content items for the History content collection. In a scenario where we need to train the system on a new content collection with a new taxonomy of tags, this is not a completely impractical level of effort to require before the system begins to provide meaningful results. Again, it's important to keep in mind that the results reported here are based on a relatively basic approach. It is realistic to expect that with a more complete analysis of the data set a far more refined solution could be developed and with it higher levels of accuracy.

[23] 5% equates to Business Studies: 23 samples, History: 48 samples, Science: 36 samples
[24] 75% equates to Business Studies: 352 samples, History: 720 samples, Science: 544 samples
[25] 85% equates to Business Studies: 399 samples, History: 816 samples, Science: 616 samples

7. Other frameworks and platforms

Following on from our experimentation with Azure Machine Learning Studio we then looked at other available Machine Learning frameworks so that we could investigate the practical implications of using these tools. We looked at 2 additional ML frameworks for running our experiments. Our approach was to take the best configuration that we developed through our experimentation and try to reproduce it using the Python programming language and some of the many Machine Learning and data science libraries that are for available (including TensorFlow, Numpy, and Pandas). Following this, we investigated the ML tools and capabilities provided by Google Cloud.

7.1. Python

In this section, we will be looking at replicating our best experiment with the Machine Learning Studio, using Python, and more specifically TensorFlow and PyTorch, in order to get an idea of their practicality and performance in comparison to the Machine Learning Studio.

The first impression after testing the TensorFlow library is that it requires a deeper understanding of machine learning concepts (models, loss function, gradient descent). TensorFlow still offers a nice abstraction of the more complex concepts, but goes a step further in terms of implementation compared to the Azure Machine Learning studio, offering a very practical plug-and-play solution. However, TensorFlow offers machine-learning newcomers an enjoyable high-level abstraction of complex ML features while allowing machine-learning experts to customize their own algorithms in a very flexible way, with different levels of APIs.

7.1.1. "TensorFlow.estimator", the high-level machine-learning API

For the purpose of our experiments, and for a matter of time, we focused our effort on the "estimator" API of the TensorFlow library. The Estimator is a high-level API allowing developers to generate and apply machine-learning algorithms with very few parameters (Linear Regression, Neural Networks, etc…). Note that further customization is also possible.

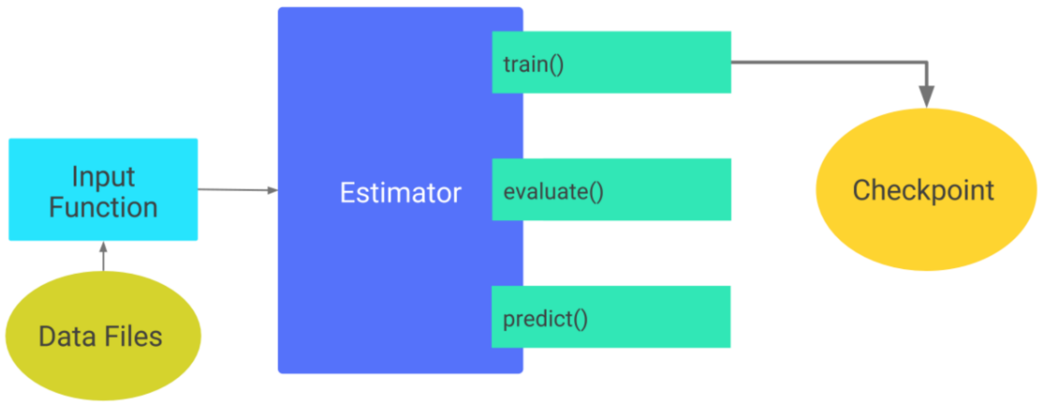


Figure 7: Functioning of the Estimator API (diagram available on TensorFlow website)

The estimator API contains 3 main methods, allowing training a predetermined Machine Learning algorithm, evaluating its success on the testing data and predicting on a new set with the trained model.

Once a model has been trained, checkpoints are generated by the .train() method, giving the opportunity for developers to reuse their previously trained model and avoid unnecessary training time (that was taking up to 30 minutes in our case). Note that using the estimator API automatically generates checkpoints (https://www.tensorflow.org/programmers_guide/saved_model). When using the lower-level API, models needs to be explicitly saved via the .save() method.

We tried to replicate with TensorFlow our previous best experiment (with 75% of training data and 25% of testing data), as realised with Azure Machine Learning Studio using Neural Networks and prior feature hashing with n-grams (2 grams). We obtained a 68% of overall accuracy with our TensorFlow model, when applied to the Business Studies collection (see the graph shown in Figure 8), 10% lower than the experiment run on Azure Machine Learning Studio. Those results may be linked to the quality of our data and the parameters of our Machine Learning algorithm, as we haven't been able to reproduce the exact same experiment we did with the Azure Machine Learning Studio.

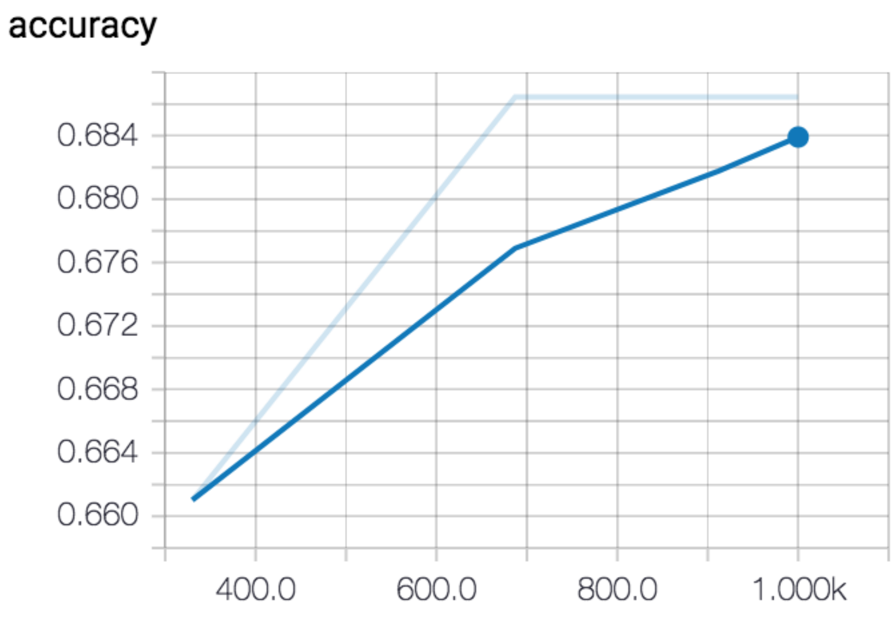


Figure 8: Overall accuracy of our TensorFlow model, based on the number of training iterations

7.1.2. Customization of your own model via the low-level API

The low-level API allows developers to create their own models from scratch, or even to customize existing Machine Learning algorithm by modifying some of their characteristics. This requires a much deeper understanding of Machine Learning and of the model you're trying to customize/create not to mention a higher level of comfort with software engineering. Even with this in mind the use of such lower level tools does require a substantial enough amount of additional effort. Simply extracting the data from a data source such as a CSV file or database and carrying out some basic processing would involve a degree of effort. Essentially every step in the pipeline from raw data to predictions requires being able to process and store the data efficiently before you can even consider training a ML model. Similarly testing the model and evaluating the results would involve more effort.

Several tutorials are available on the TensorFlow website, offering an easy way to learn for beginners with the tools although again familiarity with Python or at least programming languages in general would be a prerequisite.

7.2. Google Cloud Machine Learning Engine

After experimenting the TensorFlow library, we decided to look into the Google Cloud Machine Learning Engine. The first observation is that it is not as easily accessible and intuitive as Microsoft Azure Machine Learning Studio, allowing you to create experiments by assembling blocks. With Google Cloud ML Engine, experiments can be run programmatically via a previously created virtual environment, in the console. For the most part Google provide a computing environment that has been optimised for ML in which to run systems that have been previously created, for example on a local workstation. The implication here is that the services provided by Google do not provide the same level of support for experimenting with different datasets and configurations in comparison to Azure Machine Learning Studio. In the context of a production environment this would not be an issue but as a development tool it needs consideration.

The second observation is that Google Cloud ML Engine requires not only more Machine Learning knowledge, but it also requires fundamental knowledge of Python and TensorFlow, as the engine is built on top of the TensorFlow framework[26]. Note however that the documentation provided by Google is very complete and offers lots of tutorials (with accessible data) to get started with the Engine.

As with Azure Machine Learning Studio, it is also possible to combine the Engine with other Google APIs, such as Cloud Dataflow, Cloud Storage or Cloud Datalab. This would allow the storage and use of very large datasets as well as the integration of ML models developed with Google Cloud ML Engine to be integrated into both new and existing cloud based products and services.

[26] See here the prerequisite to get started with the Google ML Engine: https://cloud.google.com/ml-engine/docs/getting-started-training-prediction

8. Conclusion and Recommendations

8.1. Summary

To gain a better understanding of Machine Learning and its relevance in edTech, we decided to apply the technology in a real-world use case. In so doing we had two key objectives, the first was to provide a better understanding of the capabilities of available ‘off the shelf’ Machine Learning tools. The second objective was to validate whether 'off the shelf' Machine Learning tools could be used to tag (classify) eLearning content slices using a taxonomy of tags that do not generally appear in the content slices themselves. We identified the Almanac personalised learning demonstrator as an interesting use case that shared many commonalities with other contexts of use for ML. Another advantage of the Almanac use case was the availability of a real-world dataset of educational content for schools that had previously been tagged by a subject matter expert. This dataset was kindly provided by CJ Fallon, who were also involved in the original Almanac demonstrator project. In this use case we had more than 2 different tags (classes) and as such, our problem belonged to the multiclass classification. For our purposes, we trained several Machine Learning models, with different Machine Learning algorithms for multiclass classification, with the aim of helping us to predict which class a certain piece of text belongs to. We first started by experimenting with 4 different multiclass classification algorithms (Neural Network, Logistic Regression, Decision Forest and Decision Jungle) on raw text without any prior analysis or processing. We also investigated the amount of data needed to train each model (50%, 75%, 85% of the dataset), using the remainder of the data to test our model. These experiments were divided into 3 batches: we started our experiments based on the title of each piece of content, then on the content itself and finally on both features combined. For those initial experiments, we focused on the Business Studies collection of content, from the data set provided by CJ Fallon.

We were expecting better results on the last experiment of the 3, when combining both parameters, however it appeared to be quite the opposite. We obtained better results using just the title to train the ML model, when predicting based on the title only. We deduced this phenomenon was resulting of the quality of the title itself, in many cases self-explanatory (“Keywords” or “Exercise” for example) as well as the lack of any pre-processing of the text content. The overall accuracy was still relatively poor at this stage although we did obtain results that were promising enough to encourage us to look more deeply at the dataset.

The next step was then to try to improve our results by applying more advanced data analysis techniques for text classification, such as feature extraction and hashing followed by some filtering of those features, still based on our 4 initial Machine Learning algorithms. Results improved significantly when doing so, reaching 78% of precision in our predictions (still on the Business Studies collection) using Neural Networks and combining both the text and title of pieces of content (after feature hashing and filtering). This time, combining both the title and the text turned out to offer the better results, as we initially expected. As a baseline we compared the results that we achieved with how a randomly-predicting system would perform, we evaluated 2 methods, including random guesses (based on the weight of each class in the whole dataset) and the zero-rule (predicting the class the most represented in the dataset regardless of the content to classify). This allowed us to realise the improvements of our system in comparison with those baselines, going from 31% (with the zero-rule) to 78% (with Neural Networks) on the Business Studies collection. We also tried different methods, such as prior text cleaning and slight modifications of the techniques used to extract and hash features from the text, the result being an improvement in our results up to 85% of overall accuracy when predicting on the Business Studies collection.

Finally, we looked at how well our model was performing on other collections. Therefore, we applied it to 2 other collections, Science and History. Although the results were slightly lower in comparison to those obtained for the Business Studies collection, all 3 still followed the same accuracy curve and more importantly showing promising results overall.

In addressing our second objective, to gain deeper insight into available commercial tools and services, we looked at the ML solutions provided by Google as well as some open source technologies. We tried to replicate our best experiment in these Machine Learning environments. We focused on Python (with the TensorFlow library) and Google Cloud Machine Learning Engine. Both frameworks/environments require a deeper understanding of Machine Learning models and coding skills, as opposed to the intuitive plug-and-play interface that the Azure Machine Learning Studio offers. TensorFlow provides preconceived Machine Learning algorithms, via the Estimator, that can be directly applied to our data. Those preconceived algorithms can also be modified via a lower-level API, allowing programmatic customisation. When trying to replicate one of our prior experiments, we obtained a 68% overall accuracy on the Business Studies collection, 10% lower than with the Machine Learning Studio (78%). The Google Cloud Machine Learning Engine is built on top of the TensorFlow library, and therefore requires its users to be familiar with the Python framework. The Cloud Machine Learning Engine can take any TensorFlow model and allows users to automatically scale Machine Learning applications, but does not provide any additional Machine Learning tool.

8.2. Discussion and Future Work

This work has validated the original assumption that features of short, text based, items of content, such as those used in the Almanac demonstrator, contain information that can be used to predict appropriate pedagogical tags to a significant degree of accuracy. However, two questions now remain: what is the relevance of this beyond the specific use case investigated, and how might this work be applied in the real world.

Applying ML in real world edTech applications has the potential to be an enabling technology allowing new learning experiences to be provided across all sectors and a wide range of contexts. As with many new technologies, ML does not represent a fundamentally new or different capability but rather allows existing, often manual, tasks to be carried out with new levels of scale and efficiency. Solutions that were previously too costly or time consuming to provide to learners are now viable options. Similarly, tools and technologies that were previously only available in contexts in which the cost could be justified can now applied more widely. Personalised or Adaptive Learning is one such technique which, due to its heavy reliance on content and metadata, is quite costly to deliver and still very labour intensive. However, any learning solution that has a similar reliance on content to drive the learning experience has the potential for savings to be made. Use cases in which content is being created continuously or created by the learners themselves, such as in performance support or tacit knowledge based tools in corporate environments, are just some examples.

The specific pedagogical tagging problem is a particularly challenging one as the relationship between the tags or categories and the learning content contained in the text is not obvious. This is in contrast to some of the text categorisation use cases that are addressed commercially in which there is a more obvious link, for example identifying whether a news article is about sport, entertainment or current affairs. As such, the results achieved are very promising and should be indicative of the potential that ML can play in metadata generation for EdTech solutions across various different contexts from schools to corporate learning.

When applying ML in real world applications the issue of when and how to train the underlying models is an important one. Ideally a large dataset would exist that contains information on the categorisation of content items so that the system could be trained before being put into real world use. This might be practical in some use cases, for example when a known taxonomy of categories (tags) exists but what about the use case that we have explored in this work? What if we have many, new content collections that we want to tag using a taxonomy that is specific to that collection? In this case we wouldn't have the luxury of being able to train the system in advance. By the time that we have a fully tagged content collection we no longer have a need for the recommendations that ML might provide! So what use might ML be in this case? If we consider ML as a tool for supporting subject matter experts in tagging the content rather than a fully automated replacement for the subject matter experts then there is still significant potential for efficiency and cost savings. In this scenario the subject matter expert might continue to use the same tool that they are currently using to review and tag content using a predefined taxonomy but with the aid of a ML driven recommendation system that is built into the tool. As the expert begins tagging content in a new collection the system uses those tags to train itself and can then start to make recommendations to the expert as they continue to review content. This is where the analysis of how much training data is required to achieve useful levels of accuracy comes in. We have seen that with less than 100 items of content we can begin to achieve levels of accuracy greater than 50% and with less than 250 items of content this level of accuracy increases to around 75%. This semi-automatic approach to the use of ML might also be more desirable in some cases, even though it doesn’t offer the same cost saving potential. Not only would it allow the recommendations to be reviewed and adjusted continuously but more significantly it would allow the final say to remain with expert. For some potential customers this could be important as they value the expertise of the human when considering a new product or service. The cost saving here would be in increasing the efficiency of the subject matter expert, allowing them the review far more content in a shorter space of time.

8.3. Applying Machine Learning - General Recommendations

When considering the application of Machine Learning to address the challenges that you face in your existing products and services or in the development of new solutions there are a number of key recommendations that you should follow:

- Have an understanding of the fundamentals of Machine Learning and why you believe Machine Learning might to be a good fit for your particular problem before implementing it as a solution. In some cases, applying Machine Learning **may not be the optimal solution for a given problem**[27].
- Be very **clear with the problem** you want your ML algorithm to solve by developing a **well-defined** question. Most issues with AI applications using ML are caused by **ill-posed problems**. Remember that data science can only provide answers to 5 categories of problems (see section *2- A Brief Introduction to Machine Learning* for more information).
- Generally speaking, there is **no generic answer** to Machine Learning, each problem needs to be treated individually in order to be efficient and provide proper answers.
- **Preparation** of the data is crucial (cleaning, pre-processing, hashing, etc.). Think of data as the **ingredients** required by your ML algorithm to perform[28]. Therefore, in order to generate good predictions, good-quality data is essential. The better data, the more accurate results will be. Which also means that you need to be able to generate (or have access to) data in the first place.
- Be careful with the **number of features** considered by your ML algorithm to perform its predictions, and more specifically the phenomenon called the "**curse of dimensionality**". if the amount of available training data is fixed, then **over-fitting** occurs if we keep adding features to take into account (dimensions). On the other hand, if we keep adding dimensions, the amount of training data needs to grow exponentially fast to maintain the same coverage and to avoid over-fitting.
- The **choice of the algorithm depends of the problem you need to solve** (classification, regression, anomaly detection, etc…). There is **no better or worse machine learning algorithm**, and it is directly correlated to the data you have to treat (number of samples, number of features, targeted training time, etc…) and the problem you are trying to solve.
- To experiment with ML tools, the Microsoft Azure Machine Learning Studio is very **intuitive** in that regard, allowing users to **run** several experiments, **compare** their results but also **download** experiments from the gallery, to **learn** from more experienced data scientists how to deal with a certain type of problem.

[27] https://www.wired.com/story/greedy-brittle-opaque-and-shallow-the-downsides-to-deep-learning/
[28] https://docs.microsoft.com/en-us/azure/machine-learning/studio/data-science-for-beginners-is-your-data-ready-for-data-science

9. References

- http://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/
- https://medium.com/@sifium/machine-learning-types-of-classification-9497bd4f2e14
- https://www.allerin.com/blog/machine-learning-for-anomaly-detection
- https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/
- https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/
- https://www.oreilly.com/ideas/practical-applications-of-reinforcement-learning-in-industry
- https://deepmind.com/blog/alphago-zero-learning-scratch/
- https://www.wired.com/story/greedy-brittle-opaque-and-shallow-the-downsides-to-deep-learning/

Azure Machine Learning Studio

Here is a list of the references and resources in relation with the Azure Machine Learning Studio used for the purpose of this document:

- https://docs.microsoft.com/en-us/azure/machine-learning/studio/data-science-for-beginners-the-5-questions-data-science-answers
- https://docs.microsoft.com/en-us/azure/machine-learning/studio/data-science-for-beginners-is-your-data-ready-for-data-science
- https://docs.microsoft.com/en-us/azure/machine-learning/studio/data-science-for-beginners-ask-a-question-you-can-answer-with-data
- https://docs.microsoft.com/en-us/azure/machine-learning/studio/data-science-for-beginners-predict-an-answer-with-a-simple-model
- https://gallery.cortanaintelligence.com/experiments

Machine Learning Algorithms

Resources regarding ML algorithms used for the purpose of this document. Note that both sources are part of Microsoft resources and are dealing with ML algorithms of the Azure Machine Learning Studio.

- https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-cheat-sheet
- https://docs.microsoft.com/en-us/azure/machine-learning/studio/basics-infographic-with-algorithm-examples

State of the market

Here is the list of software using Machine Learning for text classification and analysis. The following resources were used to generate the State of the market section of this document.

- https://dandelion.eu/
- https://aylien.com/text-api/
- https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/
- https://cloud.google.com/natural-language/
- https://www.textrazor.com/
- https://www.paralleldots.com/text-analysis-apis

TensorFlow

Following is a list of the resources regarding the TensorFlow framework.

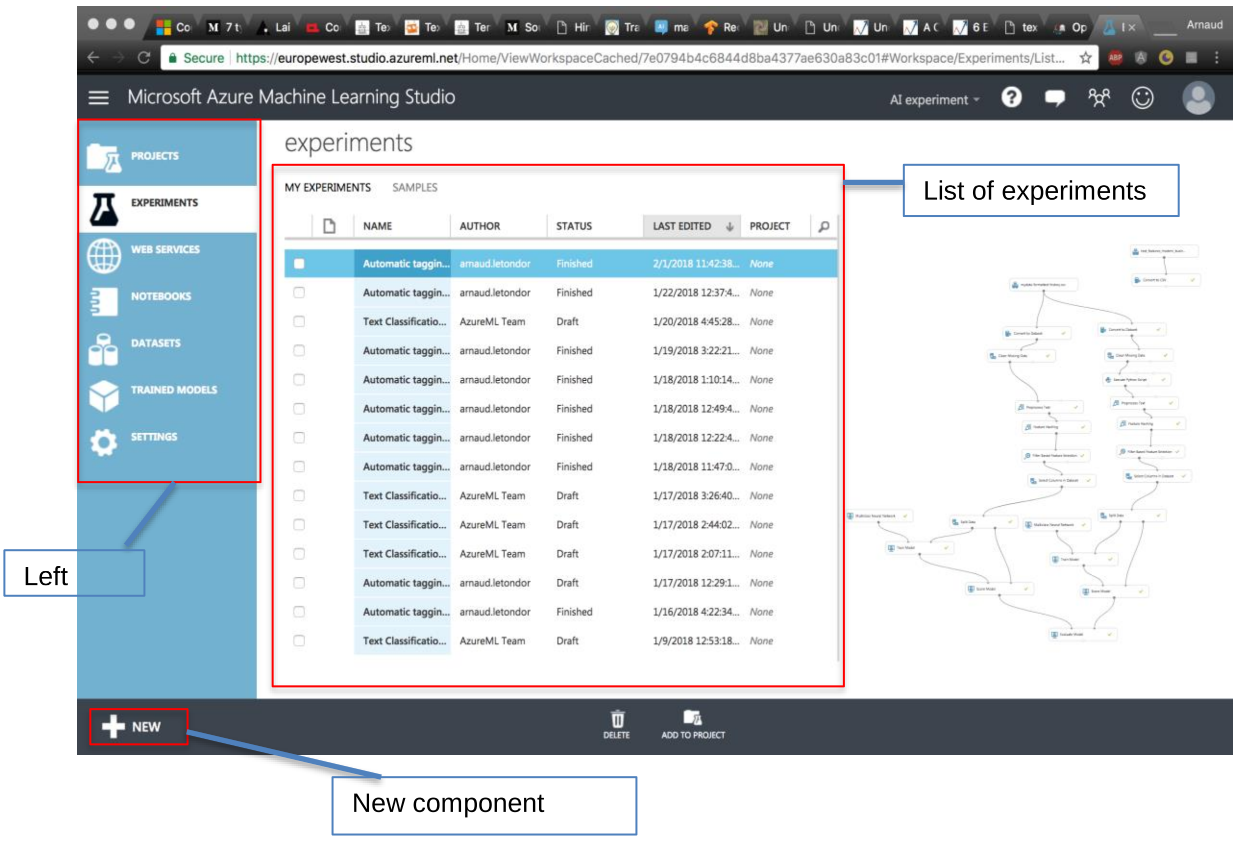
- https://www.tensorflow.org/
- https://www.tensorflow.org/get_started/premade_estimators
- https://www.tensorflow.org/get_started/checkpoints
- https://www.tensorflow.org/programmers_guide/saved_model

Google Cloud

Following is a list of the resources regarding the Google Cloud technology.

- https://cloud.google.com/ml-engine/
- https://cloud.google.com/ml-engine/docs/getting-started-training-prediction
- https://cloud.google.com/dataflow/
- https://cloud.google.com/storage/
- https://cloud.google.com/datalab/

Appendix A. Intro to Azure Machine Learning Studio



experiments

MY EXPERIMENTS SAMPLES

NAME	AUTHOR	STATUS	LAST EDITED	PROJECT
Automatic taggin...	arnaud.letondor	Finished	2/1/2018 11:42:38...	None
Automatic taggin...	arnaud.letondor	Finished	1/22/2018 12:37:4...	None
Text Classificatio...	AzureML Team	Draft	1/20/2018 4:45:28...	None
Automatic taggin...	arnaud.letondor	Finished	1/19/2018 3:22:21...	None
Automatic taggin...	arnaud.letondor	Finished	1/18/2018 1:10:14...	None
Automatic taggin...	arnaud.letondor	Finished	1/18/2018 12:49:4...	None
Automatic taggin...	arnaud.letondor	Finished	1/18/2018 12:22:4...	None
Automatic taggin...	arnaud.letondor	Finished	1/18/2018 11:47:0...	None
Text Classificatio...	AzureML Team	Draft	1/17/2018 3:26:40...	None
Text Classificatio...	AzureML Team	Draft	1/17/2018 2:44:02...	None
Text Classificatio...	AzureML Team	Draft	1/17/2018 2:07:11...	None
Automatic taggin...	arnaud.letondor	Draft	1/17/2018 12:29:1...	None
Automatic taggin...	arnaud.letondor	Finished	1/16/2018 4:22:34...	None
Text Classificatio...	AzureML Team	Draft	1/9/2018 12:53:18...	None

Figure 9: View of the Machine Learning Studio once connected, with the list of experiments

Figure 8 provides a screenshot illustrating what Azure Machine Learning Studio looks like. You can access the different menus from the left sidebar, giving you access to your projects, experiments, web services, notebooks (on Jupyter[29]), datasets, trained models and settings respectively. In the above screenshot, we visualise all the experiments of this account. Note that you can set up a free account and access all functionalities of the Azure Machine learning studio, with some restrictions in terms of computation time and number of experiments you can run. The panel on the right-hand side provides a visualisation of the currently selected experiment.

To create a new component (whether it is a new dataset, module, project, experiment or notebook), click on the "New" button, in the bottom left corner. This menu appears, and you can then create your new component. In our case, we will create a new experiment.

[29] http://jupyter.org/

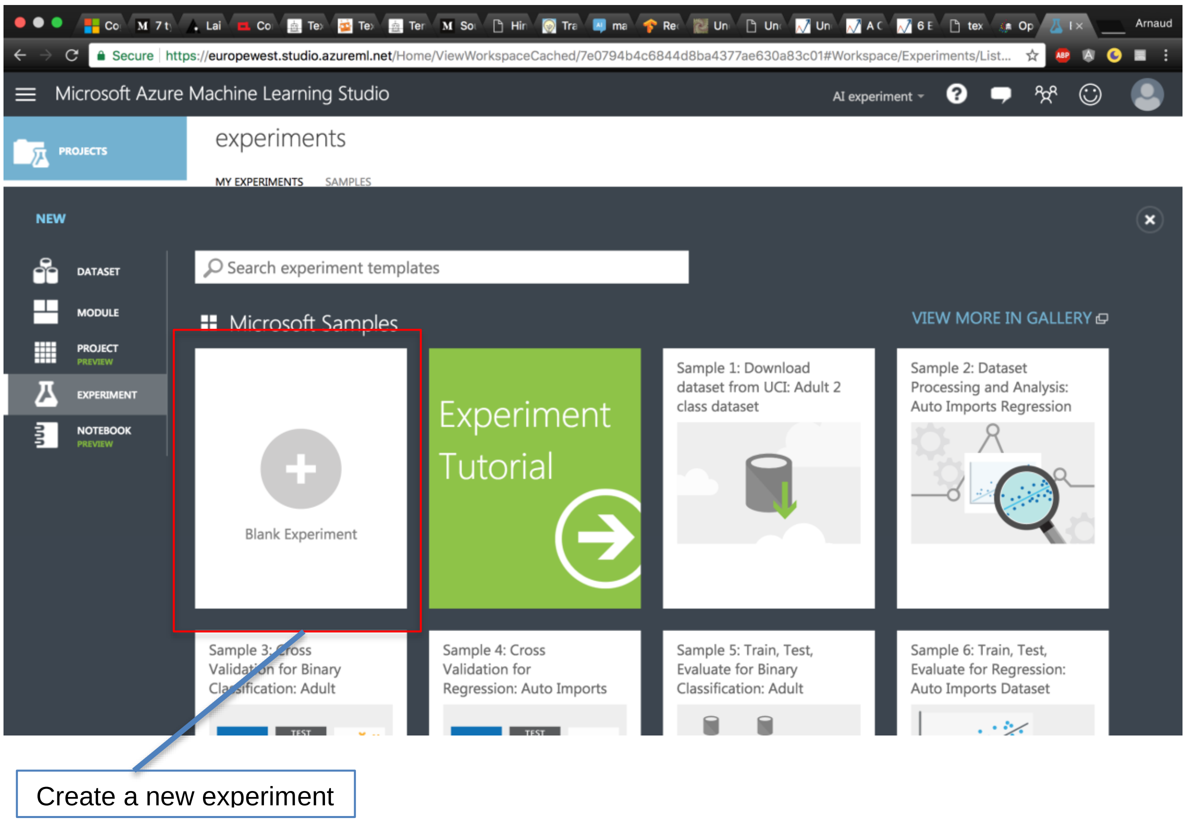

Microsoft Azure Machine Learning Studio
experiments
MY EXPERIMENTS
SAMPLES
NEW
DATASET
MODULE
PROJECT PREVIEW
EXPERIMENT
NOTEBOOK PREVIEW
Search experiment templates
Microsoft Samples
VIEW MORE IN GALLERY
Blank Experiment
Experiment Tutorial
Sample 1: Download dataset from UCI: Adult 2 class dataset
Sample 2: Dataset Processing and Analysis: Auto Imports Regression
Sample 3: Cross Validation for Binary Classification: Adult
Sample 4: Cross Validation for Regression: Auto Imports
Sample 5: Train, Test, Evaluate for Binary Classification: Adult
Sample 6: Train, Test, Evaluate for Regression: Auto Imports Dataset
Create a new experiment

Figure 10: Create a new experiment on the Machine Learning Studio

To create a new experiment, and start from scratch, users can click on the “Blank Experiment” button, as illustrated in Figure 9, which will give them an “empty sheet” (see the figure below) where they can try and experiment the different features of the tool.

Note that you can also take existing experiments from the Cortana gallery, allowing you to experiment on more experienced data-scientist’s previous models. This provides a nice way to see how the tool works, and what a regular Machine Learning solution looks like, for you to be able to create your own. When starting your own experiment, you are given access to a menu of tools (dataset, Machine Learning algorithm, training method, data-manipulation operations, etc…), broken down into categories, that you can directly drag and drop onto the sheet that represents your experiment (see below).

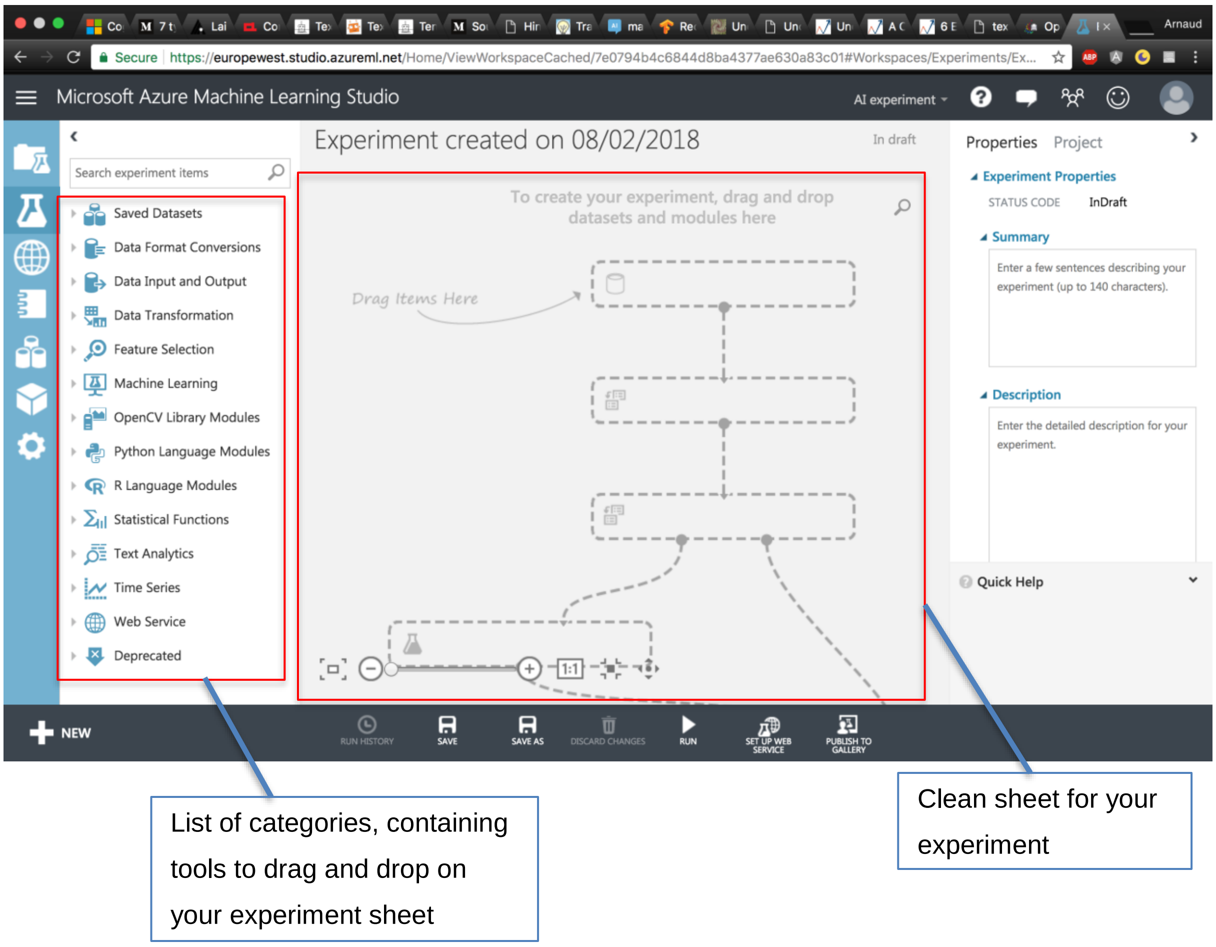


Figure 11: Experiment view on the Machine Learning Studio

Once you are familiar with the different tools that the Machine Learning Studio offers you, you can then start building your own solution (see the following figure).

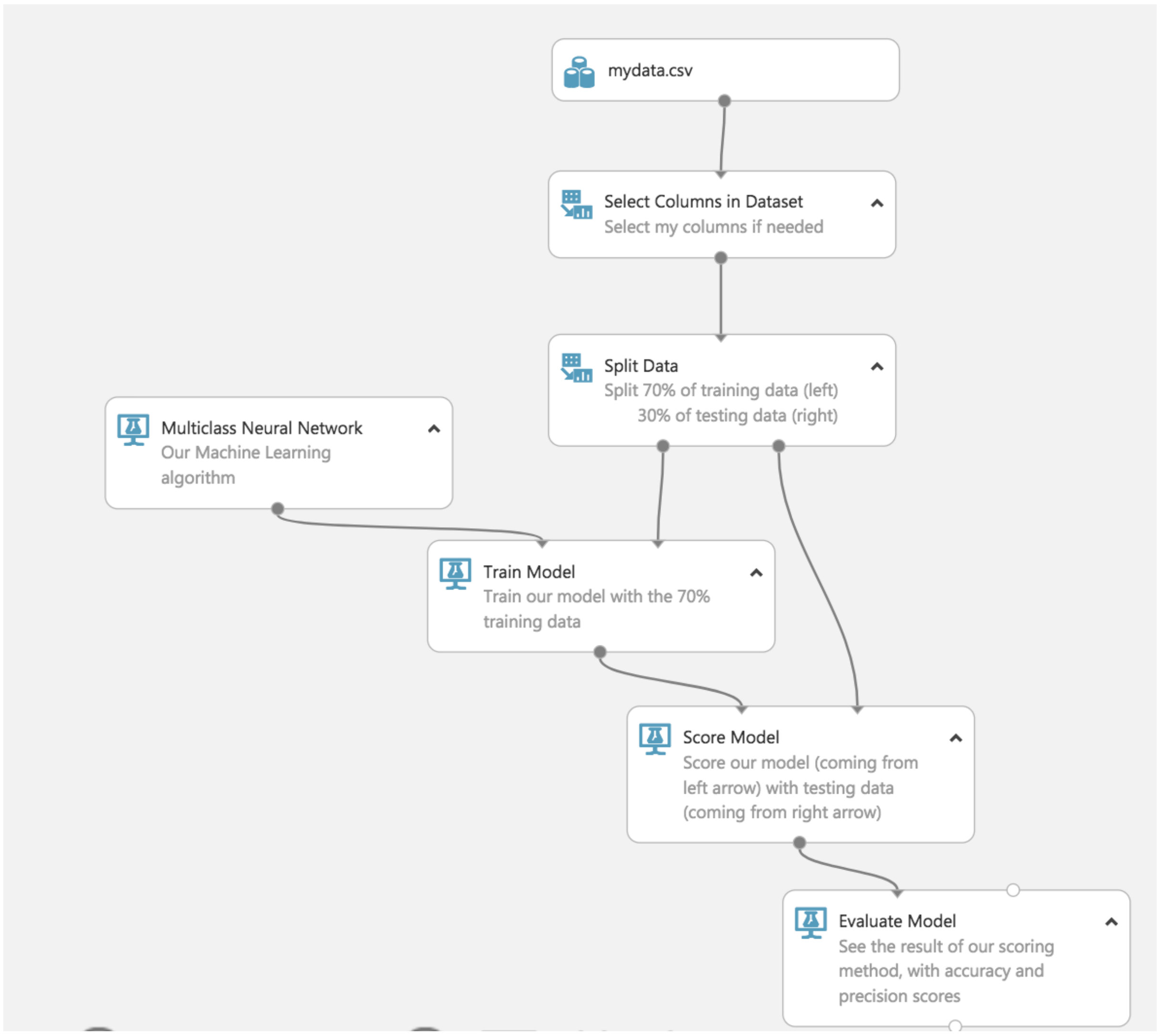
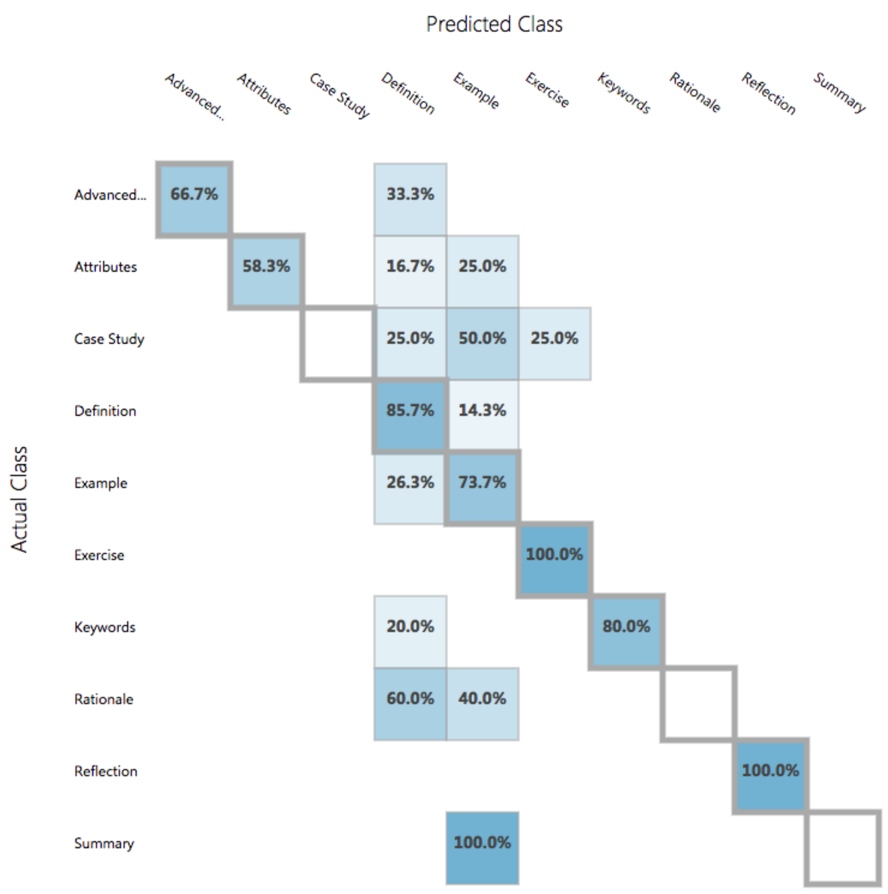

mydata.csv
Select Columns in Dataset
Select my columns if needed
Split Data
Split 70% of training data (left)
30% of testing data (right)
Multiclass Neural Network
Our Machine Learning algorithm
Train Model
Train our model with the 70% training data
Score Model
Score our model (coming from left arrow) with testing data (coming from right arrow)
Evaluate Model
See the result of our scoring method, with accuracy and precision scores

Figure 12: Simple experiment (with comments for each step) on the Machine Learning Studio

Figure 11 above represents a very basic implementation of Neural Network on a random dataset (imported with the mydata.csv block), which we select our columns from (features and target value). Once we select the data we are interested in from our dataset, we split our data into 2 distinct datasets, representing our training data (that goes into the left arrows) and our testing data (into the right arrows). Note that each block can be modified, so that we customize it according to our needs. In the case of the split, we can specify how we want to split our data (in our case, 70%-30%).
Our training data then goes into the "Train Model" block that is going to train our Neural Network algorithm (on the upper left) with that data. The result of the "Train Model" block is our own Machine Learning model, which we apply on our testing data via the "Score Model" block. Here, our model is going to perform predictions on our testing data. Once the model scoring is done, we "evaluate" our model, resulting in accuracy measurements. The result of the model evaluation is a confusion matrix

(see the section _6.4- How to read and understand a Confusion Matrix_), showing us how well our model performed on our testing data (see Figure 7 below for an example of a confusion matrix).


Predicted Class
Actual Class

	Advanced...	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary
Advanced...	66.7%			33.3%						
Attributes		58.3%		16.7%	25.0%					
Case Study				25.0%	50.0%	25.0%				
Definition				85.7%	14.3%					
Example				26.3%	73.7%					
Exercise						100.0%				
Keywords				20.0%			80.0%			
Rationale				60.0%	40.0%					
Reflection									100.0%	
Summary					100.0%					

Figure 13: Example of Confusion Matrix generated by Azure Machine Learning Studio

Note that in this introduction, we focused on creating experiments, but you can as well make them part of a project, set up experiments as web services or publish it to the Cortana Gallery, all of those actions available from the experiment edition page.

Appendix B. Initial Experiment Results

All experiments carried out on the Azure Machine Learning Studio has been reported under the form of a table, presented following the same format. Each table contains 13 columns:

Training data: This column represents the percentage (5%, 10%, 25%, …, 85%) of training data taken out of our overall data in order to train our model, which means that we split our overall data into 2 sets (if we take 75% of training data, we will have 25% of testing data, and so on). This gives an idea of the average of data required for us to train our model in the most efficient way. **Remember that our predictions** (reported in the table) **are made on our testing data**, which means that when we are training our model with 15% of our data as training data, we will have 85% of testing data to make predictions on.

Tags: One column per tag (class) of the collection Business studies. Each cell contains 2 information: the accuracy specific to each class, alongside the detailed number of slices of the given class correctly predicted out of the total number (of our testing data) of slices actually corresponding to a given class.

Overall Accuracy: The overall accuracy is a way of measuring the efficiency of our model, and represents the number of correctly predicted items out of the overall number of items. Also corresponds to the Precision.

Average Accuracy: The average accuracy is different, as it considers how often the classifier is correct, averaged per class. Note that it also takes true negative into consideration, which explains why its results are most of the time high.

Therefore, we want to optimize both overall and average accuracy scores.

Caption for each experiment table:

🟧	Accuracy between 1% and 49%
🟨	Accuracy between 50% and 74%
🟩	Accuracy between 75% and 99%
🟩	Perfect accuracy (100%)

With title & topic

Using Neural Network

Note that we will be using default Neural Networks, provided by the Machine Learning Studio, only modifying the number of iterations through our experiments. However, when applying Neural Networks (NN) outside the scope of the Azure Machine Learning Studio, it is possible to apply different kind of NNs, using different kinds of structures.

<u>Experiment 1:</u>
With 100 hidden neurons, 100 training iterations:

Table 11: Neural Network experiment (experiment 1)

Training Data	Pedagogical Tags (Classes)										Overall Accuracy	Average Accuracy
	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary		
5%	0/13	3/49 6.1%	0/14	0/85	12/74 16.2%	86/136 63.2%	0/20	2/17 11.8%	2/33 6.1%	0/5	23%	85%
10%	0/12	3/46 6.5%	0/13	14/80 17.5%	0/73	79/128 61.7%	3/20 15%	2/16 12.5%	6/30 20%	1/4 25%	26%	85%
25%	0/12	0/39	0/11	13/69 18.8%	6/61 9.8%	68/101 67.3%	17/18 94.4%	0/13	3/25 12%	0/3	30%	86%
50%	0/5	2/27 7.4%	0/6	11/43 25.6%	14/41 34.1	46/74 62.2%	13/13 100%	0/8	7/15 46.7%	0/2	40%	88%
60%	0/4	0/23	0/6	10/36 27.8%	12/31 38.7%	44/57 77.2%	9/9 100%	0/8	3/12 25%	0/2	41%	88%
75%	0/3	0/13	0/5	13/20 65%	5/21 23.8%	28/36 77.8%	5/5 100%	0/3	3/9 33%	0/2	46%	89%
85%	0/0	0/10	1/4 25%	0/16	3/17 17.6%	8/31 25.8%	0/5	0/2	0/8	0/1	12%	82%

Experiment 2:

With 100 hidden neurons, 1000 training iterations:

Table 12: Neural Network experiment (experiment 2)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
5%	0/13	3/49 6.1%	0/14	0/85	12/74 16.2%	86/136 63.2	15/20 75%	2/17 11.8%	2/33 6.1%	0/5	27%	85%
10%	0/12	3/46 6.5%	0/13	14/80 17.5%	0/73	89/128 69.5%	15/20 75%	1/16 6.3%	6/30 20%	1/4 25%	30%	86%
25%	0/12	0/39	0/11	13/69 18.8%	7/61 11.5%	78/101 77.2%	17/18 94.4%	0/13	3/25 12%	0/3	33%	87%
50%	0/5	2/27 7.4%	0/6	14/43 32.6%	14/41 34.1	47/74 63.5%	13/13 100%	0/8	6/15 40%	0/2	41%	88%
60%	0/4	0/23	0/6	13/36 36%	11/31 35.5%	44/57 77.2%	9/9 100%	0/8	3/12 25%	0/2	42%	88%
75%	0/3	0/13	0/5	8/20 40%	8/21 38%	32/36 89%	5/5 100%	0/3	3/9 33%	0/2	48%	89%
85%	0/0	0/10	1/4 25%	0/16	3/17 17.6%	3/31 9.7%	0/5	0/2	0/8	0/1	7%	81%

Best result: 78% training data, with an overall accuracy of 53%.

Using Logistic Regression

Experiment 3:

Table 13: Logistic Regression experiment (experiment 3)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
5%	0/13	0/49	0/14	0/85	0/74	136/136 100%	0/20	0/17	0/33	0/5	30%	86%
10%	0/12	0/46	0/13	0/80	0/73	128/128 100%	0/20	0/16	0/30	0/4	30%	86%
25%	0/12	0/39	0/11	8/69 11.6%	0/61	99/101 98%	17/18 94.4%	0/13	0/25	0/3	30%	86%
50%	0/5	0/27	0/6	5/43 11.6%	13/41 31.7%	67/74 90.5%	13/13 100%	0/8	2/15 13.3%	0/2	42%	88%
60%	0/4	0/23	0/6	4/36 11.1%	7/31 22.6%	54/57 94.7%	9/9 100%	0/8	1/12 8.3%	0/2	40%	88%
75%	0/3	0/13	0/5	4/20 20%	5/21 23.8%	35/36 97.2%	5/5 100%	0/3	3/9 11.1%	0/2	42%	88%
85%	0/0	0/10	0/4	0/16	1/17 5.9%	3/31 9.7%	0/5	0/2	0/8	0/1	4%	81%

Best result: 78% training data, with an overall accuracy of 45%.

Using Decision Forest

Experiment 4:

Table 14: Decision Forest experiment (experiment 4)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
10%	0/12	0/46	0/13	2/80 2.5%	0/73	114/128 89.1%	8/20 40%	0/16	0/30	0/4	29%	86%
25%	0/12	0/39	0/11	13/69 18.8%	6/61 9.8%	77/101 76.2%	17/18 94.4%	0/13	0/25	0/3	32%	86%
50%	0/5	0/27	0/6	6/43 14%	2/41 4.9%	72/74 97.3%	13/13 100%	0/8	4/15 26.7%	0/2	41%	88%
75%	0/3	0/13	0/5	8/20 40%	5/21 23.8%	32/36 88.9%	5/5 100%	0/3	2/9 22.2%	0/2	44%	89%
85%	0/0	0/9	1/4 25%	0/10	2/12 16.7%	2/23 8.7%	0/4	0/2	0/5	0/1	7%	81%

Best result: 75% training data, with an overall accuracy of 44%.

Using Decision Jungle

Experiment 5:

Table 15: Decision Jungle experiment (experiment 5)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
10%	0/12	0/46	0/13	0/80	0/73	128/128 100%	18/20 90%	1/16 6.3%	0/30	0/4	35%	87%
25%	0/12	0/39	0/11	11/69 15.9%	0/61	93/101 92.1%	17/18 94.4%	0/13	0/25	0/3	34%	86%
50%	0/5	0/27	0/6	5/43 11.6%	2/41 4.9%	68/74 91.9%	13/13 100%	0/8	3/15 20%	0/2	39%	88%
75%	0/3	0/13	0/5	4/20 20%	5/21 23.8%	35/36 97.2%	5/5 100%	0/3	2/9 22.2%	0/2	43%	88%
85%	0/0	0/9	0/4	0/10	1/12 8.3%	1/23 4.3%	0/4	0/2	0/5	0/1	3%	80%

Best result: 75% training data, with an overall accuracy of 43%.

With text & topic

Using Neural Network

Experiment 6:

With 100 hidden neurons, 1000 training iterations:

Table 16: Neural Network experiment (experiment 6)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
10%	0/13	1/41 2.4%	0/13	35/77 45.5%	0/73	59/131 45%	0/17	0/19	0/33	0/5	22%	84%
25%	0/11	0/35	0/11	19/64 29.7%	5/65 7.7%	74/106 69.8%	0/16	0/14	2/25 8%	0/5	28%	86%
50%	0/8	1/24 4.2%	0/9	13/38 34.2%	16/47 34%	51/71 71.8%	0/9	0/6	0/20	0/2	35%	87%
75%	0/4	0/13	0/4	6/21 28.6%	5/23 21.7%	30/32 93.8%	0/4	0/4	0/11	0/1	35%	87%
85%	0/1	0/10	0/1	4/11 36.4%	3/13 23.1%	19/23 82.6%	0/2	0/1	0/7	0/1	37%	87%

Best result: 85% training data, with an overall accuracy of 37%.

Using Logistic Regression

Experiment 7:

Table 17: Logistic Regression experiment (experiment 7)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
10%	0/13	1/41 2.4%	0/13	0/77	0/73	120/131 91.6%	0/17	0/19	0/33	0/5	29%	86%
25%	0/11	0/35	0/11	10/64 15.6%	0/65	103/106 97.2%	0/16	0/14	0/25	0/5	32%	86%
50%	0/8	0/24	0/9	6/38 15.8%	3/47 6.4%	69/71 97.2%	0/9	0/6	0/20	0/2	33%	87%
75%	0/4	0/13	0/4	5/21 23.8%	5/23 21.7%	31/32 96.9%	0/4	0/4	0/11	0/1	35%	87%
85%	0/1	0/10	0/1	2/11 18.2%	3/13 23.1%	21/23 91.3%	0/2	0/1	0/7	0/1	37%	87%

Best result: 85% training data, with an overall accuracy of 37%.

Using Decision Forest

Experiment 8:

Table 18: Decision Forest experiment (experiment 8)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
25%	0/11	5/35 14.3%	0/11	10/64 15.6%	5/65 7.7%	77/106 72.6%	0/16	0/14	3/25 12%	0/5	28%	86%
50%	0/8	0/24	1/9 11.1%	7/38 18.4%	3/47 6.4%	68/71 95.8%	0/9	0/6	0/20	0/2	34%	87%
75%	0/4	0/13	0/4	9/21 42.9%	9/23 39.1%	30/32 93.8%	0/4	0/4	0/11	0/1	41%	88%
85%	0/1	0/10	0/1	3/11 27.3%	3/13 23.1%	23/23 100%	0/2	0/1	0/7	0/1	41%	88%

Best result: 75% training data, with an overall accuracy of 41% (no variation at 85% of training data).

Using Decision Jungle

Experiment 9:

Table 19: Decision Jungle experiment (experiment 9)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
25%	0/11	0/35	0/11	10/64 15.6%	0/65	103/106 97.2%	0/16	0/14	0/25	0/5	32%	86%
50%	0/8	0/24	1/9 11.1%	6/38 15.8%	3/47 6.4%	68/71 95.8%	0/9	0/6	0/20	0/2	33%	87%
75%	0/4	0/13	0/4	3/21 14.3%	5/23 21.7%	31/32 96.9%	0/4	0/4	0/11	0/1	33%	87%
85%	0/1	0/10	0/1	2/11 18.2%	2/13 15.4%	23/23 100%	0/2	0/1	0/7	0/1	38%	88%

Best result: 85% training data, with an overall accuracy of 38%.

With title & text & topic

Using Neural Network

Experiment 10:

With 100 hidden neurons, 1000 training iterations:

Table 20: Neural Network experiment (experiment 10)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
10%	0/13	1/41 2.4%	0/13	36/77 46.8%	0/73	60/131 45.8%	16/17 94.1%	0/19	2/33 6.1%	0/5	27%	85%
25%	0/11	0/35	0/11	17/64 26.6%	5/65 7.7%	82/106 77.4%	15/16 93.8%	0/14	7/25 28%	0/5	36%	87%
50%	0/8	1/24 4.2%	0/9	13/38 34.2%	16/47 34%	55/71 77.5%	8/9 88.9%	0/6	5/20 25%	0/2	42%	88%
75%	0/4	0/13	0/4	8/21 38.1%	9/23 39.1%	29/32 90.6%	4/4 100%	0/4	3/11 27.3%	0/1	45%	89%
85%	0/1	0/10	0/1	5/11 45.5%	7/13 53.8%	20/23 87%	2/2 100%	0/1	2/7 28.6%	0/1	51%	90%

Best result: 85% training data, with an overall accuracy of 51%.

Using Logistic Regression

Experiment 11:

Table 21: Logistic Regression experiment (experiment 11)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
10%	0/13	1/41 2.4%	0/13	0/77	0/73	120/131 91.6%	15/17 88.2%	0/19	0/33	0/5	32%	86%
25%	0/11	0/35	0/1	10/64 15.6%	0/65	103/106 97.2%	13/16 81.3%	0/14	0/25	0/5	36%	87%
50%	0/8	0/24	0/9	6/38 15.8%	3/47 6.4%	69/71 97.2%	8/9 88.9%	0/6	0/20	0/2	37%	87%
75%	0/4	0/13	0/4	5/21 23.8%	5/23 21.7%	31/32 96.9%	4/4 100%	0/4	1/11 9.1%	0/1	39%	88%
85%	0/1	0/10	0/1	2/11 18.2%	3/13 23.1%	21/23 91.3%	2/2 100%	0/1	1/7 14.3%	0/1	41%	88%

Best result: 85% training data, with an overall accuracy of 41%.

Using Decision Forest

Experiment 12:

Table 22: Decision Forest experiment (experiment 12)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
50%	0/7	0/25	0/7	9/43 20.9%	10/38 26.3%	60/72 83.3%	9/10 90%	0/10	2/19 10.5%	0/2	39%	88%
75%	0/3	1/12 8.3%	0/4	4/21 19%	7/19 36.8%	30/36 83.3%	4/5 80%	0/5	1/9 11.1%	0/1	41%	88%
85%	0/2	0/7	0/2	1/13 7.7%	2/11 18.2%	19/22 86.4%	3/3 100%	0/3	1/6 16.7%	0/1	37%	87%

Best result: 75% training data, with an overall accuracy of 41%.

Using Decision Jungle

Experiment 13:

Table 23: Decision Jungle experiment (experiment 13)

Training Data	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
50%	0/7	0/25	0/7	9/43 20.9%	10/38 26.3%	56/72 77.8%	9/10 90%	0/10	3/19 15.8%	0/2	37%	87%
75%	0/3	0/12	0/4	3/21 14.3%	7/19 36.8%	30/36 83.3%	4/5 80%	0/5	1/9 11.1%	0/1	39%	88%
85%	0/2	0/7	0/2	1/13 7.7%	2/11 18.2%	20/22 90.9%	3/3 100%	0/3	1/6 16.7%	0/1	38%	88%

Best result: 75% training data, with an overall accuracy of 39%.

Appendix C. Round 2 Experimental Results

Table 24: Comparison of each ML algorithm with Feature hashing and filtering (text + title)

ML Algorithm (75%)	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
NN	2/3 66.7%	7/12 58.3%	0/4	19/21 90.5%	13/19 68.4%	36/36 100%	4/5 80%	0/5	9/9 100%	0/1	78%	96%
LR	2/3 66.7%	6/12 50%	0/4	18/21 85.7%	10/19 52.6%	36/36 100%	4/5 80%	0/5	9/9 100%	0/1	74%	95%
DF	2/3 66.7%	3/12 25%	0/4	19/21 90.5%	6/19 31.6%	36/36 100%	4/5 80%	0/5	9/9 100%	0/1	69%	94%
DJ	0/3	4/12 33.3%	0/4	17/21 81%	4/19 21.1%	36/36 100%	4/5 80%	0/5	9/9 100%	0/1	64%	93%

NN: Neural Network

LR: Logistic Regression

DF: Decision Forest

DJ: Decision Jungle

We notice after the first experiment using the N-grams hashing method that our model performs way better than it used to, going from 45% to 78% accuracy with NNs and from 39% to 74% with LR. Here are the same experiments, run using the same parameters for feature hashing and filter based feature selection, applied to text only and to title only.

Table 25: Comparison of each ML algorithm with Feature hashing and filtering (title only)

ML Algorithm (75%)	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
NN	2/3 66.7%	2/12 16.7%	0/4	13/21 61.9%	8/19 42.1%	36/36 100%	4/5 80%	0/5	9/9 100%	0/1	64%	93%
LR	0/3	3/12 25%	0/4	20/21 95.2%	9/19 47.4%	36/36 100%	4/5 80%	0/5	9/9 100%	0/1	70%	94%
DF	0/3	2/12 16.7%	0/4	19/21 90.5%	10/19 52.6%	36/36 100%	4/5 80%	0/5	9/9 100%	0/1	70%	94%
DJ	0/3	1/12 8.3%	0/4	18/21 85.7%	3/19 15.8%	36/36 100%	4/5 80%	0/5	9/9 100%	0/1	62%	92%

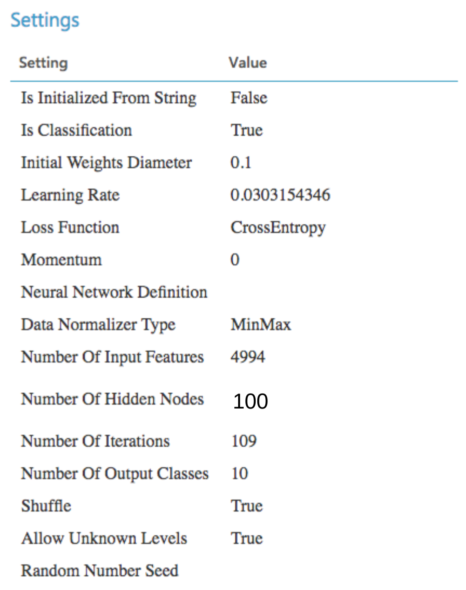
Table 26: Comparison of each ML algorithm with Feature hashing and filtering (text only)

ML Algorithm (75%)	Advanced Organizer	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary	Overall Accuracy	Average Accuracy
NN	0/3	8/12 66.7%	0/4	11/21 52.4%	14/19 73.7%	31/36 86.1%	3/5 60%	0/5	4/9 44.4%	0/1	62%	92%
LR	2/3 66.7%	4/12 33.3%	0/4	10/21 47.6%	9/19 47.4%	34/36 94.4%	4/5 80%	0/5	1/9 11.1%	0/1	56%	91%
DF	2/3 66.7%	2/12 16.7%	0/4	15/21 71.4%	7/19 36.8%	35/36 97.2%	2/5 40%	0/5	4/9 44.4%	0/1	58%	92%
DJ	0/3	3/12 25%	0/4	12/21 57.1%	2/19 10.5%	34/36 94.4%	2/5 40%	0/5	1/9 11.1%	0/1	47%	89%

We can here notice that both experiments perform better than without feature hashing and filter based feature selection, however both perform less accurately than with text and title combined. That confirms our initial assumption stating that the predictions of our model would increase when taking into account both parameters.

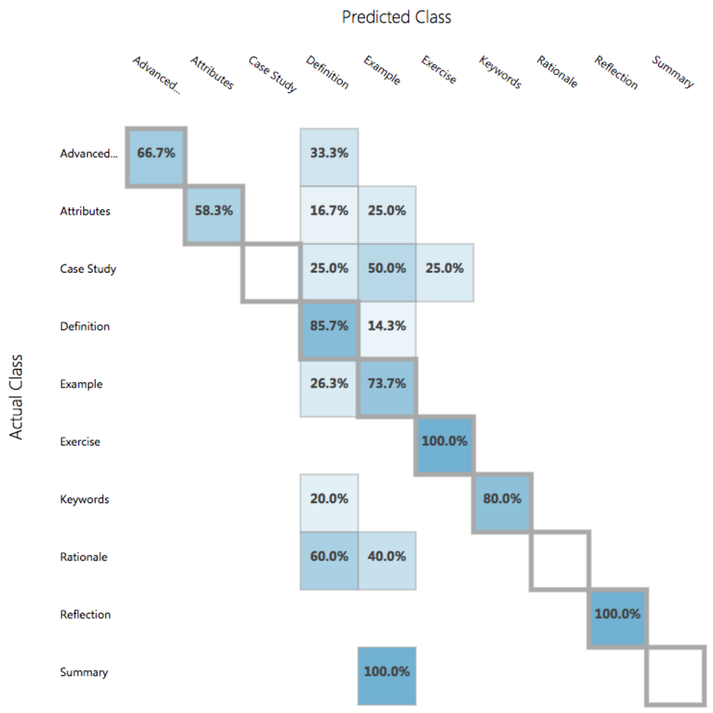
Appendix D. Experimental Result Visualisations

Neural Network Multiclass Classifier



Settings

Setting	Value
Is Initialized From String	False
Is Classification	True
Initial Weights Diameter	0.1
Learning Rate	0.0303154346
Loss Function	CrossEntropy
Momentum	0
Neural Network Definition	
Data Normalizer Type	MinMax
Number Of Input Features	4994
Number Of Hidden Nodes	100
Number Of Iterations	109
Number Of Output Classes	10
Shuffle	True
Allow Unknown Levels	True
Random Number Seed	



Predicted Class / Actual Class

	Advanced...	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary
Advanced...	66.7%			33.3%						
Attributes		58.3%		16.7%	25.0%					
Case Study				25.0%	50.0%	25.0%				
Definition				85.7%	14.3%					
Example				26.3%	73.7%					
Exercise						100.0%				
Keywords				20.0%			80.0%			
Rationale				60.0%	40.0%					
Reflection									100.0%	
Summary					100.0%					

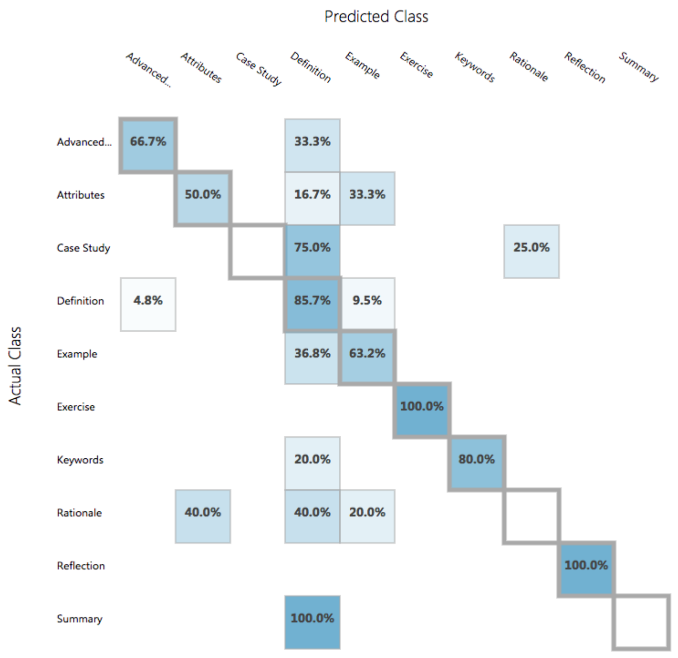
Overall Accuracy	Average Accuracy
78%	96%

Multiclass Logistic Regression Classifier

Settings

Setting	Value
Optimization Tolerance	5.472686E-06
L1 Weight	0.08110995
L2 Weight	0.195253327
Memory Size	25
Quiet	True
Use Threads	True
Allow Unknown Levels	True
Random Number Seed	

Overall Accuracy	Average Accuracy
76%	95%

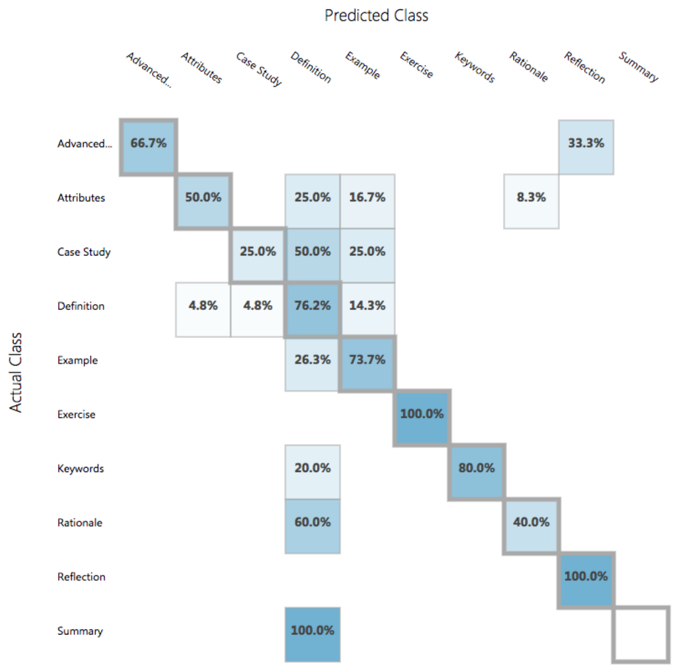


One-vs-All Multiclass (with two-class Decision tree classifier)

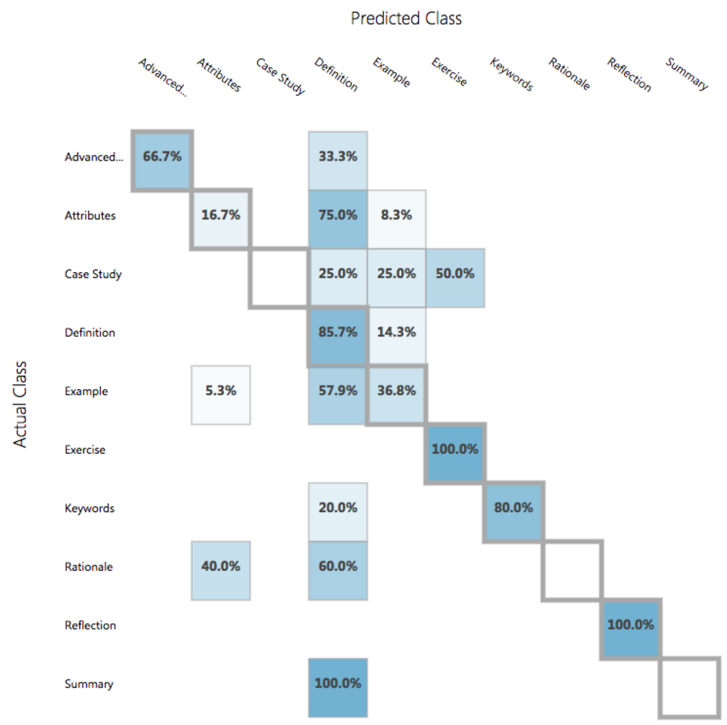
Settings

Setting	Value
Number Of Leaves	70
Minimum Leaf Instances	4
Learning Rate	0.09517171
Number Of Trees	237
Allow Unknown Levels	True
Random Number Seed	

Overall Accuracy	Average Accuracy
78%	96%

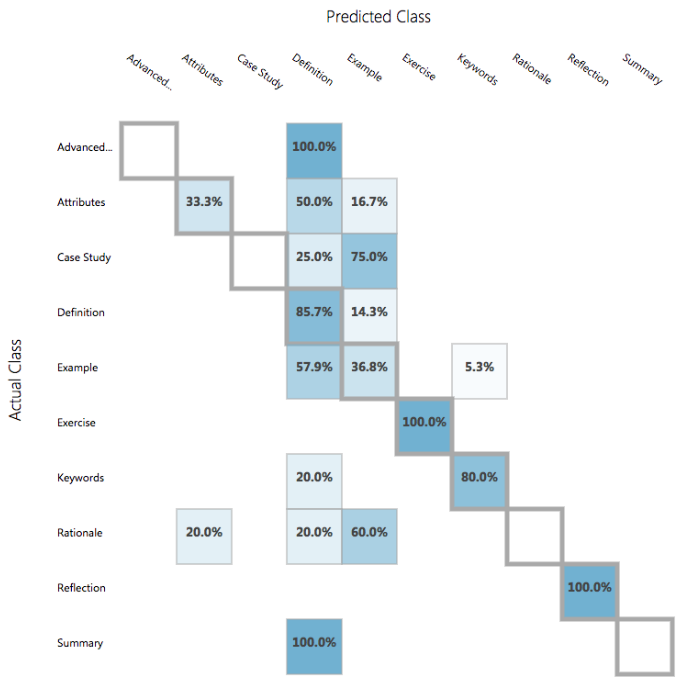


Decision Forest



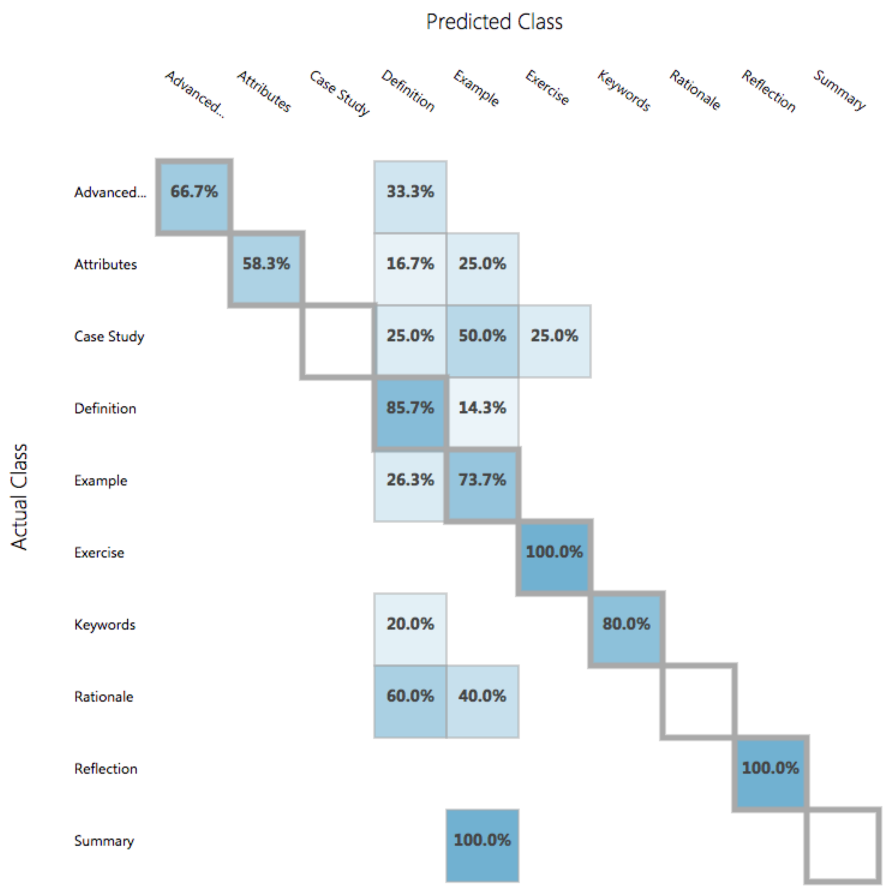
Overall Accuracy	Average Accuracy
67%	94%

Decision Jungle



Overall Accuracy	Average Accuracy
68%	94%

How to read and understand a Confusion Matrix


Predicted Class
Actual Class

	Advanced...	Attributes	Case Study	Definition	Example	Exercise	Keywords	Rationale	Reflection	Summary
Advanced...	66.7%			33.3%						
Attributes		58.3%		16.7%	25.0%					
Case Study				25.0%	50.0%	25.0%				
Definition				85.7%	14.3%					
Example				26.3%	73.7%					
Exercise						100.0%				
Keywords				20.0%			80.0%			
Rationale				60.0%	40.0%					
Reflection									100.0%	
Summary					100.0%					

Figure 14: Example of Confusion Matrix on the Machine Learning Studio

A confusion matrix is a table used to describe the performance of a classification model. The one represented above shows a case of multiclass classification (a two-class classification confusion matrix would be slightly different, but based on the same principle). The y-axis represents the "Actual Class" of the testing data, which is the value our classifier **should predict**, and the x-axis represents the "Predicted Class" of the testing data, which is the value our classifier **did predict**. Therefore, our goal here is to get the higher score on each cell of the diagonal of the table, representing the percentage of accuracy predicting each class for our testing data.

A way of reading the matrix is line by line. If we take, for example, the second line "Attributes", we can understand it this way:
"58.3% of our testing data labelled as 'Attributes' has been correctly classified as 'Attributes', whereas 41.7% has been wrongly predicted as 'Definition' (16.7%) and 'Example' (25%)."