What is “real-world” computing?

Real-world computing will extract meaning from highly complex data. It is important to define precisely what we mean by “complex data”. We distinguish three degrees of complexity in digital data, according to how difficult it is to extract meaning from the data by computer:

1. *Structured text*. This data consists of symbols from a well-defined alphabet organized in well-defined structures, and whose semantics is easily determined from the data itself. Typical examples are spreadsheets and databases. The complete meaning of the data can be internalized by a computer with straightforward parsing techniques.
2. *Free-form text*. This data consists of symbols from a well-defined symbolic alphabet organized in well-defined structures, but whose semantics is not easily determined from the data itself. The semantics may be too complex or may not be well-defined. Determining it may be possible with a lot of computation, or it may be unknown how to determine it. Data mining, which is a branch of machine learning, is a form of meaning extraction from free-form text.
3. *Complex data*. This data is similar to category (2), except that the symbols are more fine-grained and it requires much more processing to determine their semantics. For example, free-form text can consist of Web pages, whereas complex data may consist of digitized photographs or videos. Determining the meaning of part of a photograph, such as face recognition, is much harder than extracting meaning from a textual Web page.

These three degrees of digital data differ mainly in quantitative properties. From degree (1) to (3), they increase successively in abundance and in difficulty of meaning extraction. Existing software mostly handles degree (1); degrees (2) and (3) are mostly transmitted and copied, with creation and meaning extraction done by human users. Some software exists to extract meaning from degree (2): for example, data mining and language translation. Google's corpus-based statistical language translation is an example of meaning extraction from degree (2).

The goal of real-world computing is to address degree (3), which is the most complex but has great potential. Very little software exists for extracting meaning from degree (3), and what software does exist is limited and resource-hungry. Despite this, the processing of degree (3) would be extremely useful for human users, and it will therefore be the focus of this Flagship. Typical examples are real-time audio language translation and assistance for complex real-time human tasks. We assert that learning algorithms are on the verge of being sufficient to successfully tackle degree (3) applications. We motivate this assertion by giving some examples of the state of the art. The popular Dragon NaturallySpeaking speech recognition software handles continuous dictation on a single machine (no pauses between words at fast speaking rates). It is a practical tool to replace typing (at increased speed, since people usually speak faster than they type) [NUA2010]. A second example is Google, which announced in early 2010 that it was working on practical real-time audio language translation [GOU2010]. Prototypes of this have been done much earlier (notably by the German Verbmobil project [WAH2000]). A final example is the IRCAM Research Institute in Computational Acoustics and Music in Paris, France [IRC2010]. IRCAM has developed many industrial-quality tools for sound processing that are important building blocks in a real-time audio translation application.

References

[GOU2010] Chris Gourlay. “Google leaps language barrier with translator phone,” The Times, Feb. 7, 2010 (available on the Web).
[IRC2010] IRCAM (Institut de Recherche et Coordination Acoustique/Musique), www.ircam.fr.
[NUA2010] Nuance Communications. Dragon NaturallySpeaking speech recognition software, www.nuance.com, 2010.
[WAH2000] Wolfgang Wahlster (ed.). *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer-Verlag, 2000.

Resource amplification through elastic computing

Complex processing of real-world data is only possible with enormous computing resources. We can obtain these resources through the novel property of *elasticity*. Elasticity is the ability to quickly ramp up and down resource usage on demand. In today's computing infrastructure, elasticity is provided primarily by *cloud computing*, hosted in data centers. Applications that process complex data can absorb very great computing and storage resources. This can be not only because of the problem complexity (e.g., NP-completeness), but simply because of the sheer amount of work needed to store and process large quantities of data.

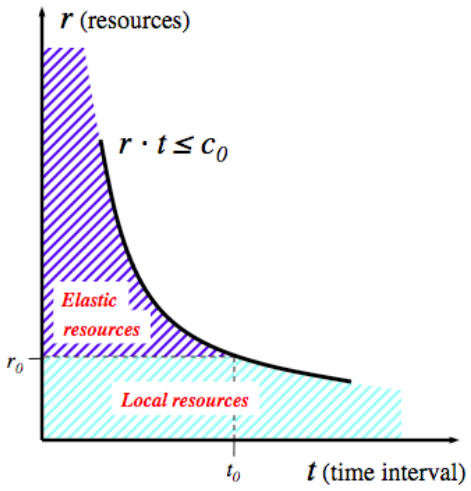

Figure 1: Additional resources available because of elasticity

By using elasticity, enormous resources can be made available to applications for short time periods. This is economical because these resources are amortized among all applications and their users. On a cloud, the run-time cost of an application is proportional in a first approximation to the product of the amount of resources (computing and storage) and the time that they are needed. Figure 1 shows what this implies. For an application whose resources have total cost c_0, the lightly shaded area shows the different combinations of resources r and time interval t that are possible when running on a single computer. The resources are limited by r_0, which is the maximum that the computer can provide. If the application runs on a cloud, then the darkly shaded area becomes available: the cloud can provide enormously more resources with cost limited to c_0 as long as the resources r are used for a limited time t, according to the formula $r \leq c_0/t$. The darkly shaded area is the focus of the Flagship. In principle, the amount of resources that are available is unlimited, if they are needed for a sufficiently short time. In practice, the maximum is limited by what the cloud is able to provide according to the application's service contract. In today's clouds this can go up to about one thousand with the appropriate contract. In tomorrow's elastic infrastructures, the limit should be determined by what applications need.

The ability to increase resource usage for the same cost, as long as the resources are used for just a short time, is very important for real-world computing applications. They will need many resources both in the learning and query phases (as explained in the next section). Elastic computing has the potential to provide these resources practically and economically. However, there are many practical limitations of current elastic infrastructures (primarily clouds): they are slow in ramping up and down resources according to need, they are limited in how far they can ramp up (taking minutes instead of, e.g., fractions of a second to ramp up), and they have other problems (e.g., security, fault tolerance, bandwidth, latency). These are strong limitations that will take many years to overcome. That is why it is important to start working on them right away.

Elastic computing: clouds and beyond

Real-world computing will succeed or fail depending on the abilities of the elastic infrastructure

that supports it. In this section we briefly recapitulate the state of the art for elastic computing and define its main properties. Today's main source of elastic computing is *cloud platforms*. A cloud is a form of client/server with novel properties that derive from its large scale. Cloud computing uses the memory and processing power of a large number of computing nodes gathered together in facilities called data centers and linked through high performance networks. Cloud users have at their disposal considerable computing resources that are both flexible and modestly priced. For example, many Web applications execute on a cloud instead of on client machines. Cloud computing has three properties that distinguish it from other forms of client/server computing [EGR2010]:

1. *Virtualization*: The ability to run applications in customized environments that are insulated from the underlying hardware. Virtualization greatly simplifies software installation and maintenance. Current virtualization techniques have a performance penalty, but they are still practical for many applications such as enterprise computing and support for small, networked devices such as mobile phones.
2. *Scalability*: The ability to provide almost any amount of computing and storage resources. This is possible because of the large size of the data center. Because of their size, data centers have an economy of scale. The Berkeley report measures them as five to seven times cheaper than enterprise installations [ARM2009].
3. *Elasticity*: The ability of an application to quickly ramp up and down resource usage on demand. Because of this ability, cloud usage is metered: the user pays only for what is actually used, with almost no entry threshold. Furthermore, the actual cost of resources is low because of the economy of scale of the data center and the amortization of its cost over all users.

Of these three properties, the true game changer is *elasticity*. It enables a whole new class of applications that were not possible before, namely applications that need large computing and storage resources for short time periods [VAN2010]. Previously such applications could not be run since the resources were simply not available. On a cloud, the resources can be requested quickly through the elastic computing mechanisms, and released when they are no longer needed (up to the practical limits of the cloud implementation).

Clouds are currently hosted in data centers, but this will quickly become inadequate to meet the demands for real-world computing. Fortunately, Internet resources outside of data centers dwarf the largest clouds by three orders of magnitude: in Jan. 2010 there were 800,000,000 Internet hosts [ISC2010] versus less than 1,000,000 hosts in the biggest cloud [COS2009]. We therefore predict that most Internet hosts will eventually acquire cloud-like abilities (the three properties mentioned above) and be federated into peer-to-peer clouds, to support the Internet's ever-increasing appetite for data-intensive applications. Medium- and small-sized clouds requiring modest investments will complete the picture. One of the goals of the Flagship is to catalyze this transition to a fully elastic Internet.

References

[ARM2009] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. "Above the Clouds: A Berkeley View of Cloud Computing," UC Berkeley, Technical Report UCB/EECS- 2009-28, Feb. 10, 2009.

[COS2009] Paolo Costa. "The Hitchhiker's Guide to the Data Centers Galaxy," Microsoft Research Cambridge, 2009.

[EGR2010] European Commission Expert Group Report. *The Future of Cloud Computing: Opportunies for European Cloud Computing Beyond 2010*. Version 1.0. Eds.: Keith Jeffery, Burkhard Neidecker-Lutz.

Rapporteur: Lutz Schubert. Jan. 2010.

[ISC2010] Internet Systems Consortium, Inc. "ISC Domain Survey," www.isc.org, 2010.

[VAN2010] Peter Van Roy. "Scale and Design for Peer-to-Peer and Cloud," Invited talk at TTI Vanguard Conference *Matters of Scale*, July 20-21, 2010, London, UK. See www.info.ucl.ac.be/~pvr/TTIVanguardPVR.pdf.

The space of machine learning applications

In real-world computing, we target applications that process complex real-world data to perform semantically significant operations at a level close to human cognition. This processing will be done by future extensions of machine learning algorithms. In this section, we classify both existing and future machine learning applications in a novel way that highlights the potential gains to be had from real-world computing. *Machine learning* (ML) is the discipline that studies how to program computers to evolve behaviors based on example data or past experience [ALP2010]. The results of machine learning are directly useful in many ways: to optimize performance, to improve intelligence, to predict the future, and to aid understanding. ML is the primary technique to solve problems for which we do not have explicit algorithms. Often it is not possible to write down an explicit algorithm because the problem is simply too large or too complex. This is usually the case for processing of complex data to extract useful high-level information, such as audio language translation mentioned before. With machine learning, programmers no longer have to laboriously write code to process data, but the computer itself writes the code according to a learning algorithm. This greatly increases programmer productivity as well as allowing programmers to tackle real-world computing problems.

The general approach is to suppose that the data has regularities that fit a given general model (the *inductive hypothesis*, also called the *inductive bias*) and to determine the best fit of the data to the model. Machine learning comprises both determining the right model and calculating the parameters for the best fit in this model. The model can define a simple classification (buy or sell a stock) or a complex behavior (how to play a game). The power of machine learning, like the power of the scientific method, depends crucially on the choice of the model. If it is properly chosen, the model is not a limitation. On the contrary, a good model can give essential insights on the phenomena.

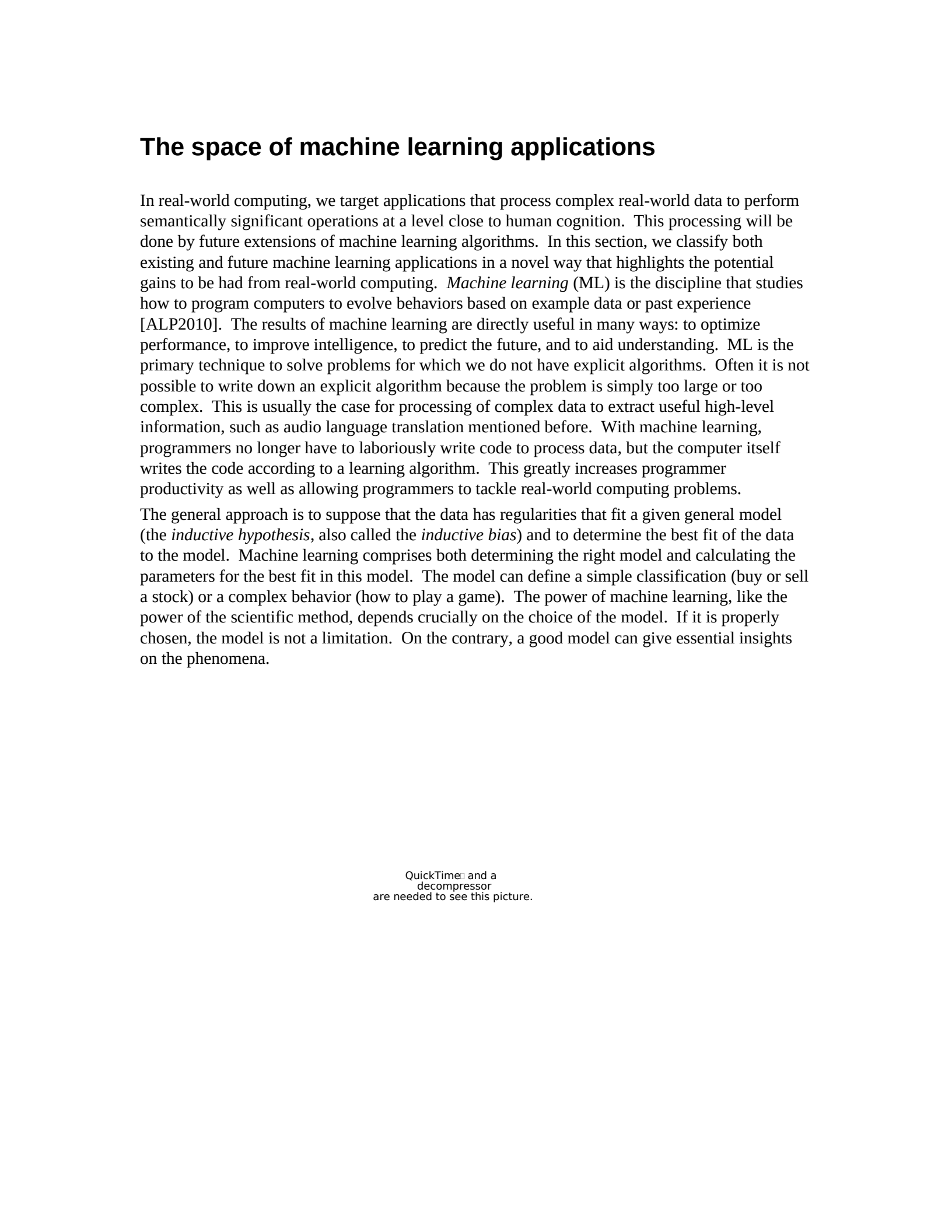

Figure 1: The space of machine learning algorithms

The space of machine learning algorithms is vast. It covers well-known kinds of applications such as Web search, corpus-based translation, recommendation systems, and computer games. But it is by far not limited to this. Most applications that require large computational and/or storage resources can be considered of this form. To achieve good results, these algorithms usually run in two phases: a *learning phase* whenever new information comes in, and a *query phase* when being used. We note that the learning phase is sometimes (partially) done by the manufacturer, such as in speech recognition systems. Figure 1 classifies algorithms in a three-dimensional space, depending on whether the learning and query phases have high elastic

degrees of interactivity: whether the algorithm is used for one-shot queries, with one-way streams, or in conversations (the darker or redder applications are more interactive). The figure shows some existing applications in this space. We are careful to separate the learning phase of an implemented algorithm from improvements in the algorithm itself. The latter can be considered as a form of "second order" learning. But it is much more unpredictable than a programmed learning phase: it is tantamount to predicting the future of the discipline of machine learning. Therefore Figure 1 gives only the learning phase of an existing algorithm without taking algorithmic improvements into account.

The highly elastic resource requirement combines two simpler requirements: it means that the application needs many resources for a short time. It is not enough for the application to need high resources if there is no constraint on time, or for the application to run in a short time if it does not need many resources. Both of these cases are easily handled by systems of modest size. The difficulty is when the two requirements are combined. For example, weather forecasting needs an enormous quantity of computing and storage resources, and it must give a result in time for users to profit from the prediction.

We extend the learn/query space with a third dimension related to time, namely the degree of interactivity. For our purposes, we distinguish three common degrees of interactivity that have a significant effect on the algorithms used for the applications. In a *one-shot query*, there is a

u

single learning phase (or several widely separated learning phases) and queries that interrogate the application in between learning phases. A typical example of a one-shot query is a data processing application with a deadline: for example, media data that needs lots of resources for a one-off format transformation. In a *one-way stream*, the learning phase is continuous (information coming in and being incorporated in a stream) and the query phase consists of a set of queries, each of which uses the information that has been assimilated up to that point. A typical example of a one-way stream is Google Search, if we consider that the search database is principally affected by changes in the Web and not by the queries themselves. In a *conversation*, there is a two-way interaction between the application and the user such that the user's queries can strongly affect the future behavior of the application and vice versa. A typical example of a conversation is a *game*. This includes two-player games like chess and also multi-user role-playing games such (where the conversation is now between *n* players and the application).

References

[ALP2010] Ethem Alpaydın. *Introduction to Machine Learning*. Second Edition. MIT Press, 2010.