KTH Information and
Communication Technology

Data-Efficient Deep Learning for Independent Binary Outputs

Exploration of importance-weighted active learning, ensembling, joint training and class imbalance correction to reduce label complexity and training time in affiliate e-commerce product clasification

MATTIAS ARRO

Master's Thesis at KTH Information and Communication Technology
MSc Data Science (EIT Digital track)

Academic Examiner: Magnus Boman
Academic Supervisor: Jim Dowling
Industrial Supervisor: Abubakrelsedik Karali

2018

Abstract

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Keywords: Deep learning, machine learning, neural networks, active learning

Referat

Denna fil ger ett avhandlingsskelett. Mer information om LaTeX-mallen finns i dokumentationen till paketet.

Acknowledgment

..... London, UK, March 28, 2018
Mattias Arro

Contents

Abbreviations

LSTM	Long Short Term Memory
NN	Neural Network
RNN	Recurrent Neural Network

Chapter 1

Introduction

Machine learning (ML) has become hugely successful over the past few years, with a lot of this newfound interest, hype, and hysteria directed at neural networks and deep learning. This focus is not unfounded - deep learning approaches continue to break benchmarks in core machine learning research areas such as computer vision [cite], speech recognition [cite], and an increasing number of natural language processing tasks [cite NMT]. Deep learning has also revolutionised reinforcement learning, achieving superhuman performance in complex games and driving vehicles in real-world situations. There are even limited results in beating human at highly uncertain games with various actors such as [Texas Holdem] poker [cite]. Understandably, academics and industry are are scrambling to apply panacea to their field.

While their superficial resemblance to natural brains might be a good topic for a sensationalist article, artificial neural networks are simply layers of non-linear transformations capable of learning complex mappings from multidimensional inputs to (usually multidimensional or structured) outputs. The building blocks of neural networks are relatively simple and the algorithms for training them are universal; this makes neural networks applicable to a variety of domains, and opens up fascinating opportunities of multimodal and transfer learning. Being able to arbitrarily increase model complexity by increasing its depth or width allows the same neural network approximate more complex functions. Increased model complexity increases training time and requires more labeled training data, yet deep models are somewhat unique in that their performance continues to increase when the dataset size increases, whereas the benefits of more data taper off for many other kinds of models. This does not automatically mean neural networks can only be used with large datasets - after all a single layer neural network can be equivalent to a logistic regression model - but that model complexity should increase with the amount of data available.

It is not immediately obvious which kind of model should one use for a given task and dataset. A data scientist can consider the following factors: how many labelled and unlabelled data points do we have, how much do we value predictive

performance, interpretability, and whether we want to do some transfer learning or joint training with the models. Labelling is often expensive, so in many real world use cases a lower label complexity (number of labels needed to obtain the desired accuracy) is preferred over slightly better performance. Deep learning seems to have a disadvantage in this aspect, but as we see in section [ref] in cases where unlabelled data is also abundant, semi-supervised and generative models can overcome low label complexity while increasing computation time. In cases where the ability of neural networks to learn features that can be used in downstream models (e.g. features learned for classification could later be used as part of a recommended system) this increased computation and engineering complexity might be justifiable.

In this thesis, we explore three orthogonal ways of efficiently learning on a proprietary dataset for product classification, where initial labels are abundant but noisy. We first evaluate different kinds of models (shallow, deep, tree-structured, convolutional, recurrent) that are trained on different modalities / input dimensions (image, text, categorical, numerical) of the same data. After determining the performance of these baseline models, the strongest models are trained as an ensemble that outperforms each individual baseline model. Finally we fine-tune the ensemble via an active learning strategy described in section [ref], where a combination of uncertainty and disagreement sampling determines a batch of products to be labeled for the next training iteration. This overcomes the noisiness and incompleteness of the initial labels without requiring much manual labeling.

1.1 Problem

The client company gathers data from various affiliate networks (that in turn give their data from various retailers) and displays the data on their online store. There are millions of products belonging to roughly 800 categories, and categories follow the usual nested tree structure. The incoming data is extremely noisy and inconsistent: what kind of data is stored in what kind of feature column varies across affiliate networks, across retailers within an affiliate network, and the data within a retailer can have lots of missing values, noisy text, missing images, etc. There is currently a rule-based system for assigning products to categories: all products matching a condition (e.g. title contains the word “trousers”) will be assigned to that category, i.e. categories are not mutually exclusive. This way of categorising products works relatively well on some categories, but such a rule-based system has several drawbacks: these rules are cumbersome to define, their evaluation is manual, they failed to match a large fraction of products that in principle should be in a given category, it is hard to trace back the rule that caused a false positive, and such rules are limited to textual data.

The client needs a classification system of binary independent outputs to replace the old way of categorising products. The system should be able to learn from the output of the old system, and if possible produce models that can be used in downstream tasks such as recommend systems and product similarity models (transfer

learning). The highest priority is low label complexity, beating requirements for high accuracy and transfer learning capabilities. The system should be robust to noisy inputs; data preprocessing should not consider the idiosyncrasies of each affiliate network. There is an additional feature the client requires: given a product image, the visitor should be shown products that are visually similar.

1.2 Purpose

The academic purpose of this work is to (1) assess the relative strengths of different kinds of models and their ensembles, and (2) to determine whether an active labelling strategy reduces label complexity on a real-world dataset. Analogously, the commercial purpose is to (1) obtain a model with powerful predictive capabilities, and to (2) reduce costs by using an efficient labelling strategy, and (3) obtain a high-quality product similarity score.

1.3 Goals

The goals of the work, in chronological order, is to:

- Use a pre-trained 2-dimensional convolutional neural network (2D CNN) to extract features for an approximate nearest-neighbour search of visually similar products.
- Build an interface for subjectively evaluating the visual similarity algorithm.
- Train baseline model to reproduce the behaviour of the rule-based system.
- Build an interface for labelling products and obtain small dataset with ground truth labels (further referred to as the "ground truth dataset").
- Train a number of different models on the rule-based labels. Evaluate these models on the rule-based test set as well as the ground truth dataset.
- Train a selection of models that had good performance as an ensemble, preferring model diversity over good performance. Evaluate this model on the rule-based test set as well as the ground truth dataset.
- Implement the active labelling mechanism defined in [ref] and fine-tune the ensemble (that is pre-trained on rule-based labels) in 10 labelling rounds.
- Document the results as well as the technical architecture and workflow.

1.4 Hypotheses

The following hypotheses were postulated prior to running most experiments[1]:

[1] the Wide&Deep baseline model had been trained on rule-based labels

1. Pre-trained 2D CNNs perform reasonably for visual item similarity, but fine-tuning might be needed.
2. Linear models are relatively good at predicting higher-level categories, worse at predicting lower-level categories.
3. Deep models for histogram / tf-idf inputs do not improve substantially on linear models.
4. Shallow models with embedding inputs generalise better to the ground truth data, and deep models with embedding inputs generalise slightly more.
5. XGBoost is the best performing model for textual and categorical data.
6. Convolutional neural networks (CNNs) that are pre-trained on Imagenet data are consistently good predictors of clothing products, yet failed to accurately predict categories such as technology.
7. Fine-tuning CNNs will give little, if any, performance improvement due to inefficient training data.
8. Ensembling gives better performance than any individual model.
9. A meta-learner with a few layers obtains better results than ensembling by averaging.
10. Active learning substantially decreases label complexity. This will be evaluated subjectively, given the difficulties outlined in section [ref].

How these hypotheses were evaluated is described in section 2.4.

1.5 Ethical Consideration

Given how pervasive machine learning is becoming across areas of society, it is good that discussions are happening about safety, transparency and bias in machine learning systems that have the ability to affect lives. The worst case scenario for inaccurate, opaque or biased product classification is embarrassment for the client company, therefore ethical considerations not of much concern for the client company. It is still worth reminding the reader, who might use some of the ideas in this work, that machine learning models always contain some kind of bias: from the data users as input (the way it is gathered, what is gathered), the way the data is processed, and the kinds of models used.

Already at this early stage of machine learning adoption there are cases where blackbox algorithms have been used to decide decisions that severely influence persons life - predicting reoffending probability of convicts to determine whether they are given parole or not - with terrible results. The model that was touted to be and objective substitute to a subjective judge has simply learnt that biases inherent to the data (the prejudices and racism of the judges at the time): it consistently underestimated reoffending rate of white convicts and overestimated the reoffending

rate for black convict [cite]. Even though the data did not contain explicit information about race, the algorithm was capable of implicitly detecting grace through some other variable that was correlated with race. Most data scientists will never build models with such severe consequences, but even more subtle decisions made autonomously by algorithms can prolong the inequalities of our society by consistently favouring certain characteristics such as race, gender, place of origin, etc. Prime examples are algorithms that determine whether one gets a mortgage or not, what the premium on their insurance would be, whose CV gets shortlisted for a position, and so on.

1.6 Sustainability

The reality of any e-commerce company is that increase revenue almost by definition means more waste and increased carbon released into the atmosphere as a result of production and transport. Efforts to mitigate these unwelcome side effects can be successful at a government level, though the author firmly believes there ought to be stronger intergovernmental regulation to tax carbon emissions and the consumption of materials, as currently there is absolutely no incentive for retailers or consumers to reduce waste, and few incentives to recycle it. The best a data scientist working for a retailer can hope for is poor performance of his models, which would not lead to increased sales.

1.7 Delimitations

1.8 Outline

Chapter 2

Method

Several hypotheses were tested in this work. Section gives an overview of the input data. Section 2.2 describes the technical set up required for running all these experiments and deploying the model to production before delving into the specific experiments and their evaluation methods in sections [ref].

The experiments were conducted in four distinct stages:

- Training a baseline model on the rule-based labels to get a sense of the difficulty of this problem. Since there is no fundamental difference in the way this model was trained and evaluated compared to the other classifiers, this is described in section 2.3.4 with the others. After this step, in ground truth dataset was obtained.
- Training the independent classifiers to determine best performers (2.3.4).
- Training an ensemble of the independent classifiers (2.3.2).
- Training up to 10 iterations of active learning on the strong predictor (2.3.3).

2.1 Data

config file for how to preprocess data selecting top_k for each column input representation: 1-hot, k-hot, tfidf, embedding

2.1.1 Category Structure

2.2 System Architecture

The following technologies were used to build the system which had to interact with existing services at the client company:

- Apache Airflow (AF) - a Python framework for defining workflows of long-running tasks and dependencies between these tasks.

- TensorFlow (TF) - ML framework for Python, capable of defining many kinds of models as a computation graph, and executing this graph locally or in a distributed manner.
- ML Engine (MLE) - a GCP service for running TensorFlow models (training, hyperparameter tuning, inference).
- Apache Beam - a data processing engine akin akin to Apache Spark and Apache Flink.
- Dataflow - a GCP service for executing Apache Beam workloads.
- Tensorflow Transform - a Python library with a small set of operations for data preprocessing that can run inside a TensorFlow graph as well as an Apache Beam pipeline.
- Google Cloud Storage (GCS) - Google Cloud Platform (GCP) object storage similar to Amazon S3.
- ElasticSearch (ES) - a NoSQL database with powerful full-text search and querying capabilities.
- RabbitMQ - a message queue, used for transferring data among our microservices (using the Logstash adapter, that can read from and write to (among other things) ElasticSearch and RabbitMQ).
- Flask - a simple backend web framework for Python.
- Node.js - a JacaScript backend web framework.
- React.js - a JavaScript front-end framework for JavaScript.
- Redux - a framework for persisting user interface (UI) state and application data in single page applications.
- GraphQL - a query language for building flexible APIs

Figure 2.1 shows the how data is passed between the main services, and how services and technologies interact.

All product data is stored in ElasticSearch (ES): the rule-based labels, the predictions of the ML system, and evaluation metrics from various train runs. ES is accessed from the public web application via GraphQL and the ML administration web UI (further referred to just as web UI). The web UI was initially built with Flask and React by the author as a quick way to get insight about the model, and then re-written as a more feature-rich version by an employee of the client company with Node.js, GraphQL, React and Redux. Data is pulled into the ML pipelines by dumping the results of an ES query to a local file, which is uploaded to GCS. Updates to the ES index are not done directly, since indexing the updated products is computationally expensive; instead, updates are put on a RabbitMQ queue, which is consumed by Logstash, which updates products in ES at a rate that will not overburden the servers.

All ML training and prediction happens in a batch-oriented way, encapsulated

as Airflow pipelines. Each pipeline is a directed acyclic graph of tasks, where a task can be a shell command or Python function; a pipeline defines dependencies of task execution, which allows us co-ordinate a series of operations that could be executed locally inside the Docker container running AF, or remotely such as in a GCP service. A typical pipeline dumps data locally, uploads it to GCS, schedules a Dataflow job to preprocess data, polls the Dataflow service until the Dataflow job is finished, schedules an ML engine job, polls the MLE service until it has completed, and runs an update process that reads the predictions and evaluation statistics from GCS and sends the updates to the RabbitMQ queue. Reading updates and sending these to RabbitMQ is done in a parallelised manner (using multiprocessing), since the update process is bottlenecked due to network latency as well as computing the appropriate category tree for each product (explained in 2.1.1).

There are two types of Dataflow pipelines: preprocessing and similarity pipelines. Omitting various details, dead-ends and workarounds that were needed due to prior system architecture changes[1], the pipelines had the following tasks:

- **Preprocess pipeline.** It loads the products dumped from ES (as JSON) and preprocesses data according to a configuration file (see section 2.1). All fields are cleaned of obvious noise and superfluous whitespace. Numerical fields are normalised, histograms are computed for categorical variables. Text fields are tokenised, tokens are counted and used for computing TF-IDF values. This tokenisation is handled by TensorFlow Transform: it keeps the top k tokens (or categorical values) for a categorical (or text) column, and maps these to integer indices. Which tokens (or categorical values) map to which indices determines the 1-hot input encoding of the data (or similar, see section 2.1), and will be different every time data is pre-processed again. This token-to-index mapping is encoded in a *transform function* that is saved to GCS at the end of the pipeline, and is used by a TensorFlow model to convert raw text inputs to a sparse inputs.
- **Visual similarity pipeline** -

[1] which were completely reasonable at the time, when the ML system was not a consideration

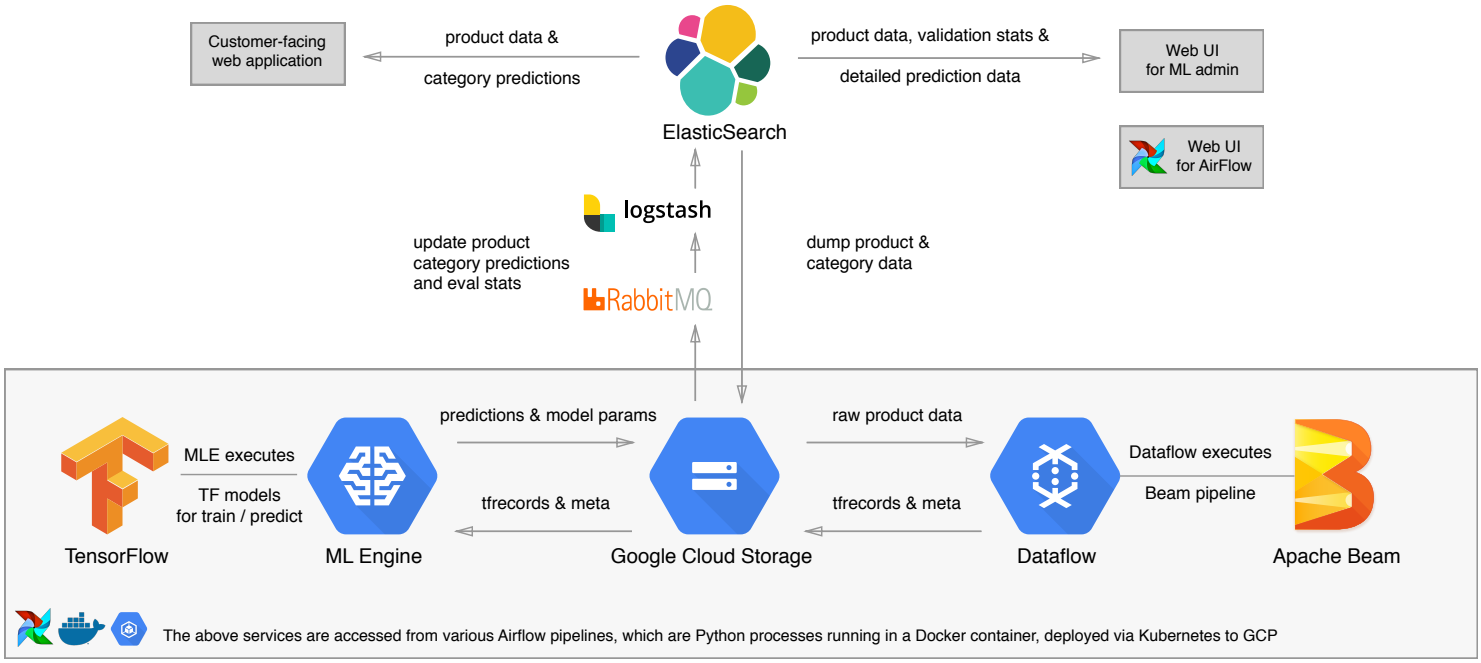

Figure 2.1. High-level system architecture of the ML pipeline

2.3 Experiments

2.3.1 Independent Models

2.3.2 Ensembling

2.3.3 Active Learning

2.3.4 Visual Similarity

2.4 Evaluation

At the beginning of running all these experiments, the dataset was divided into development and test set (90/10%). The development set was used for

we can evaluate the performance of the model on three types of datasets:

- the test or validation set as labelled by the rule-based system (referred to as "rule-based test/validation set"),
- the ground truth dataset gathered before running most experiments,
-

Chapter 3

Discussion

Chapter 4

Conclusion

References

Declaration

I hereby certify that I have written this thesis independently and have only used the specified sources and resources indicated in the bibliography.

London, UK, March 28, 2018

.....
Mattias Arro

Appendix A

RDF

And here is a figure

Figure A.1. Several statements describing the same resource.

that we refer to here: A.1