Large-scale machine learning

Syllabus

1. Introduction to large-scale machine learning
2. Distributed optimization
3. Distributed representation learning
4. *To be defined!*

Each session will combine theory and practice.

Today

1. Introduction to large-scale machine learning
   a) What is large-scale machine learning
   b) Large Scale ML frameworks and paradigms
   c) Large Scale ML on your machine

Next week: Practical session!

What is large-scale machine learning…

… and why does it matter?

Big data and the four Vs

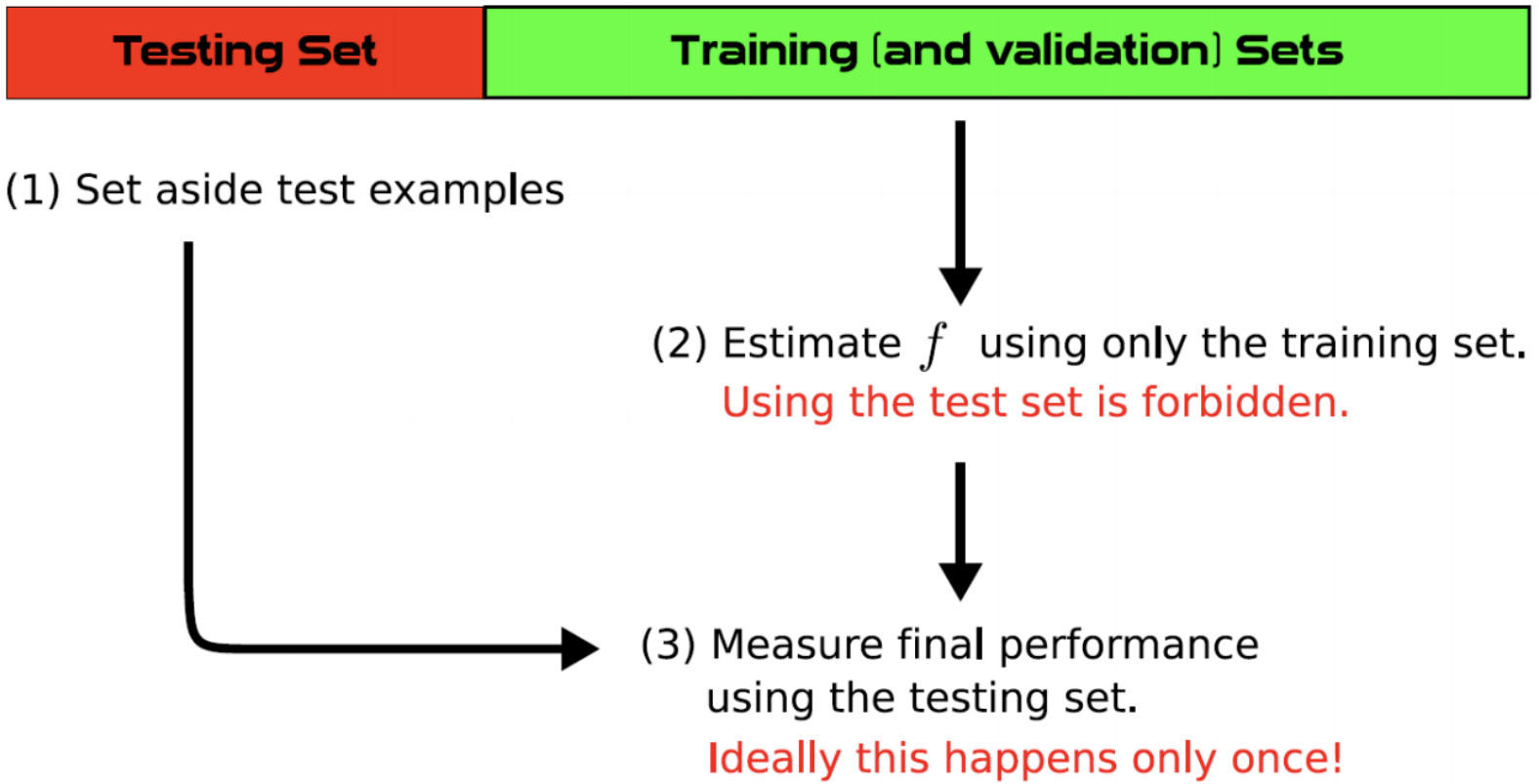
Volume (data size)
Velocity (speed of change)
Variety (different forms of data source)
Veracity (uncertainty of data)

Kaggle challenges are a sample from the real-world

The fundamental hypothesis of machine learning

Credits: Large-scale machine learning Revisited, by Leon Bottou, *Big Data: theoretical and practical challenges Workshop*, May 2013, Institut Henri Poincaré [link]

independent and identically distributed data



- This experimental paradigm has driven machine learning progress.
- The essential assumption is that training and testing data are exchangeable, e.g., follow the same distribution.

model selection tradeoffs

Approximation

- We cannot search f^* among all possible functions.
- We search instead $f^*_{\mathcal{F}}$ that minimizes the expected risk $E(f)$ within some richly parameterized family of functions $\mathcal{F}$.

Estimation

- We cannot minimize $E(f)$ because the data distribution is unknown.
- We minimize instead the empirical risk $E_n(f)$

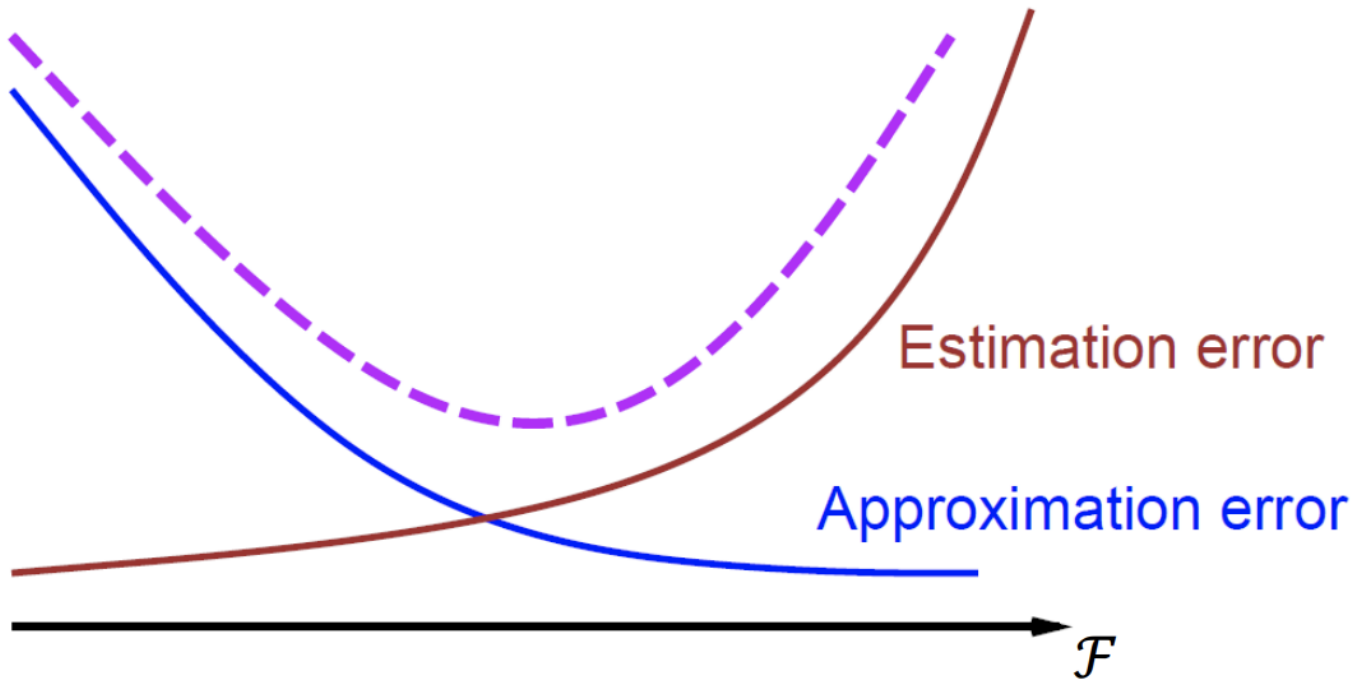
$$E_n(f) = \frac{1}{n} \sum \ell(f(x_i), y_i)$$

model selection tradeoffs

$$E(f_n) - E(f^*) = \left(E(f_F^*) - E(f^*)\right) + \left(E(f_n) - E(f_{\mathcal{F}}^*)\right)$$

Approximation Error

Estimation Error



How complex a model can you afford with your data?

learning with approximate optimization

Computing $f_n = \arg\min_{f \in \mathcal{F}} E_n(f)$ is often costly.

Since we already optimize a **surrogate** function
why should we compute its optimum f_n exactly?

Let's assume our optimizer returns $\tilde{f}_n$
such that $E_n(\tilde{f}_n) < E_n(f_n) + \rho$.

For instance, one could stop an iterative
optimization algorithm long before its convergence.

error decomposition

$$
\begin{aligned}
E(\tilde{f}_n) - E(f^*) &= E(f^*_{\mathcal{F}}) - E(f^*) && \text{Approximation error} \\
&+ E(f_n) - E(f^*_{\mathcal{F}}) && \text{Estimation error} \\
&+ E(\tilde{f}_n) - E(f_n) && \text{Optimization error}
\end{aligned}
$$

Problem:

Choose $\mathcal{F}$, n, and ρ to make this as small as possible,

subject to budget constraints $\left\{ \begin{array}{l} \text{max number of examples } n \\ \text{max computing time } T \end{array} \right.$

small scale versus large scale

Beyond informal definitions…

Small scale learning problem

- We have a small-scale learning problem when the active budget constraint is the number of examples n.

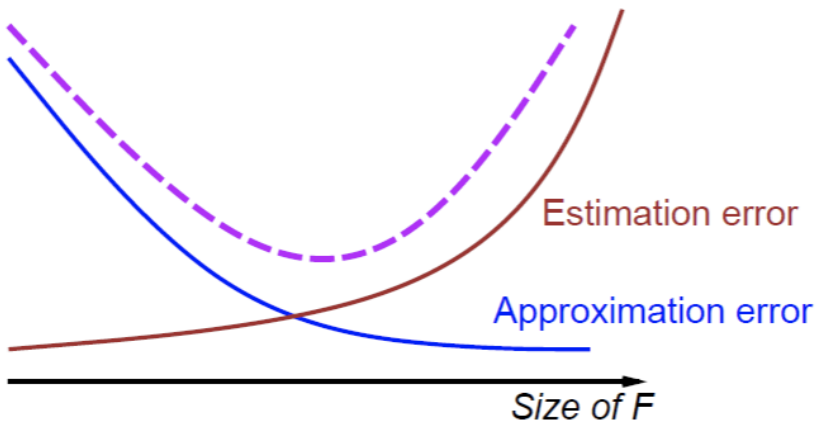
Large-scale learning problem

- We have a large-scale learning problem when the active budget constraint is the computing time T.

small-scale learning

The active budget constraint is the number of examples.

- To reduce the estimation error, take n as large as the budget allows.
- To reduce the optimization error to zero, take $\rho = 0$.
- We need to adjust the size of $\mathcal{F}$.



See Structural Risk Minimization (Vapnik 74) and later works.

large-scale learning

The active budget constraint is the computing time.

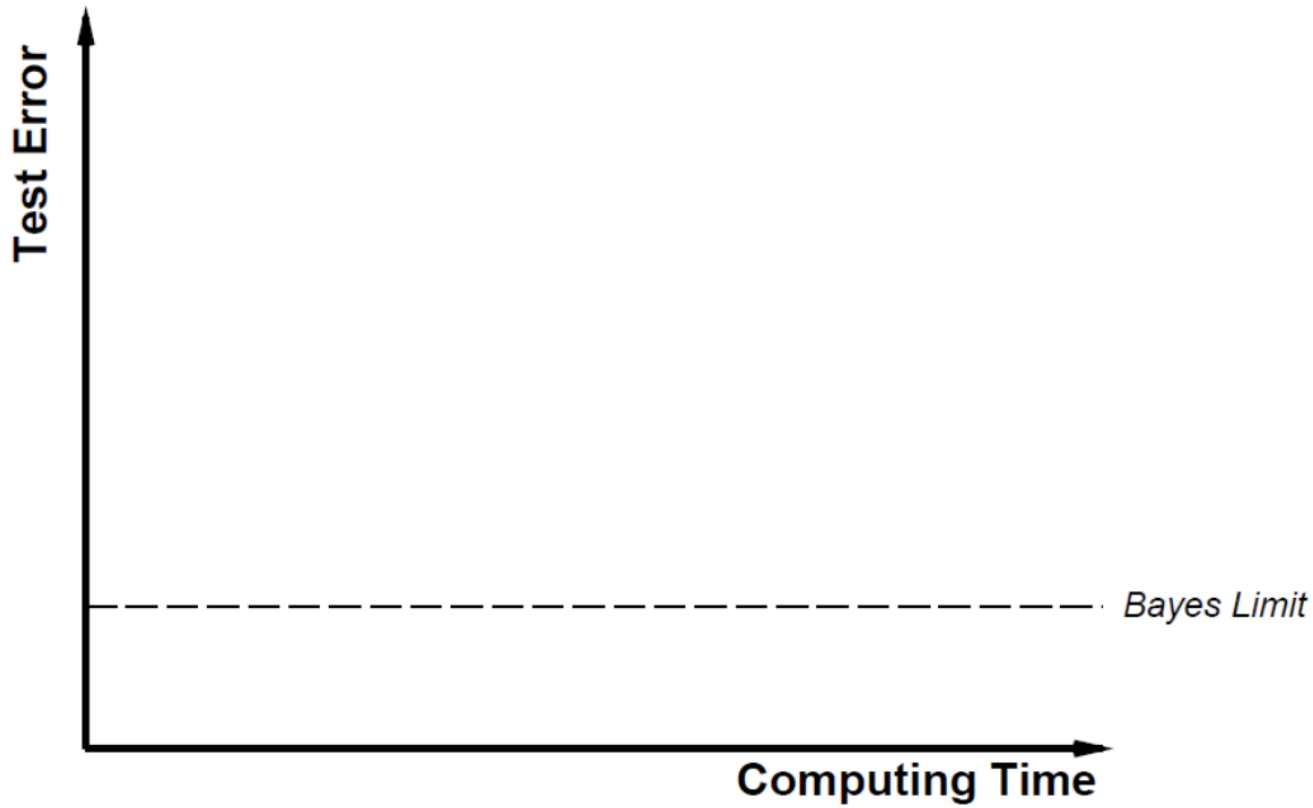
- More complicated tradeoffs.
  The computing time depends on the three variables: $\mathcal{F}$, n, and ρ.

- Example.
  If we choose ρ small, we decrease the optimization error. But we must also decrease $\mathcal{F}$ and/or n with adverse effects on the estimation and approximation errors.

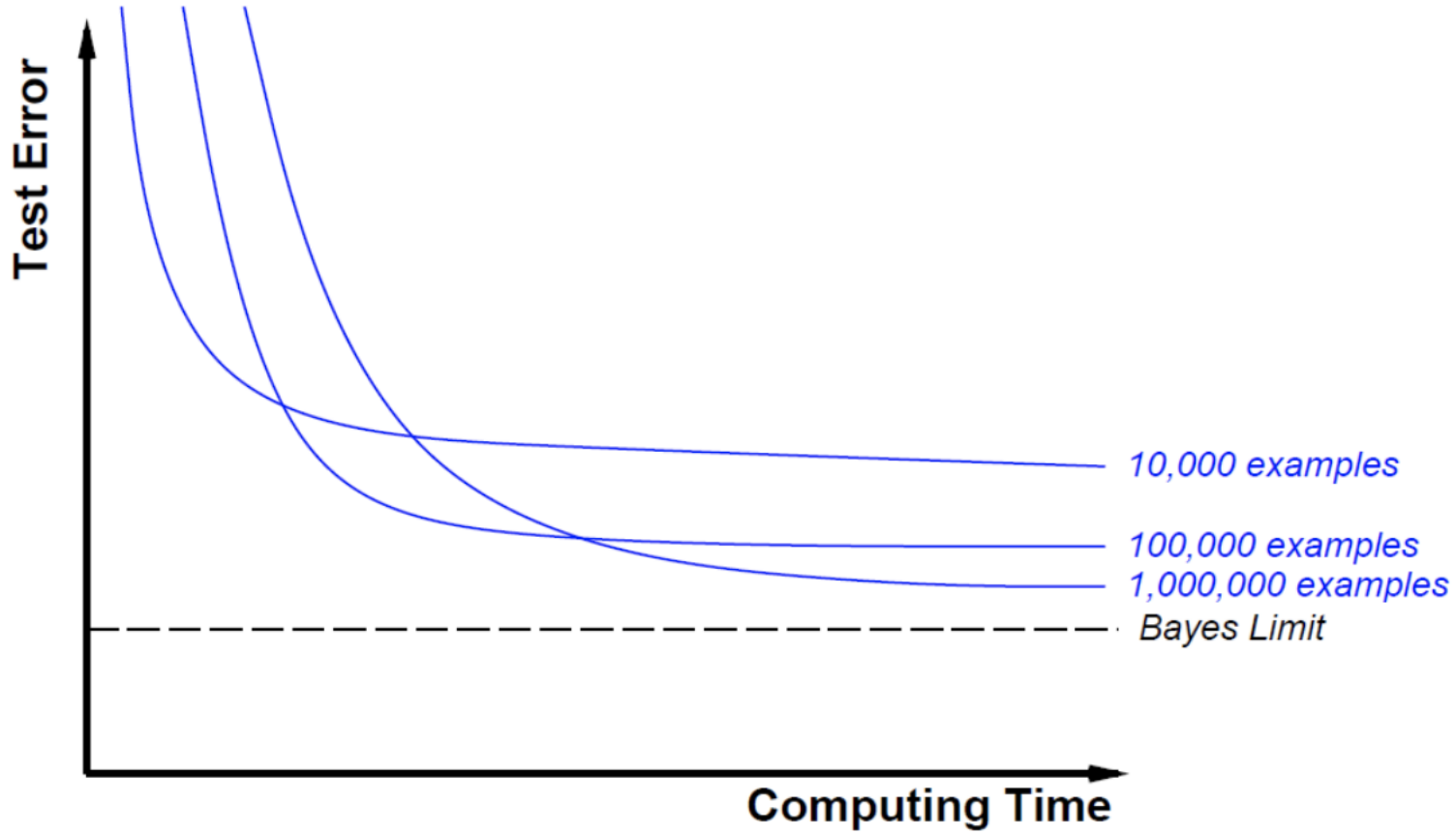
- The exact tradeoff depends on the optimization algorithm.

- We can compare optimization algorithms rigorously.

test error versus training time

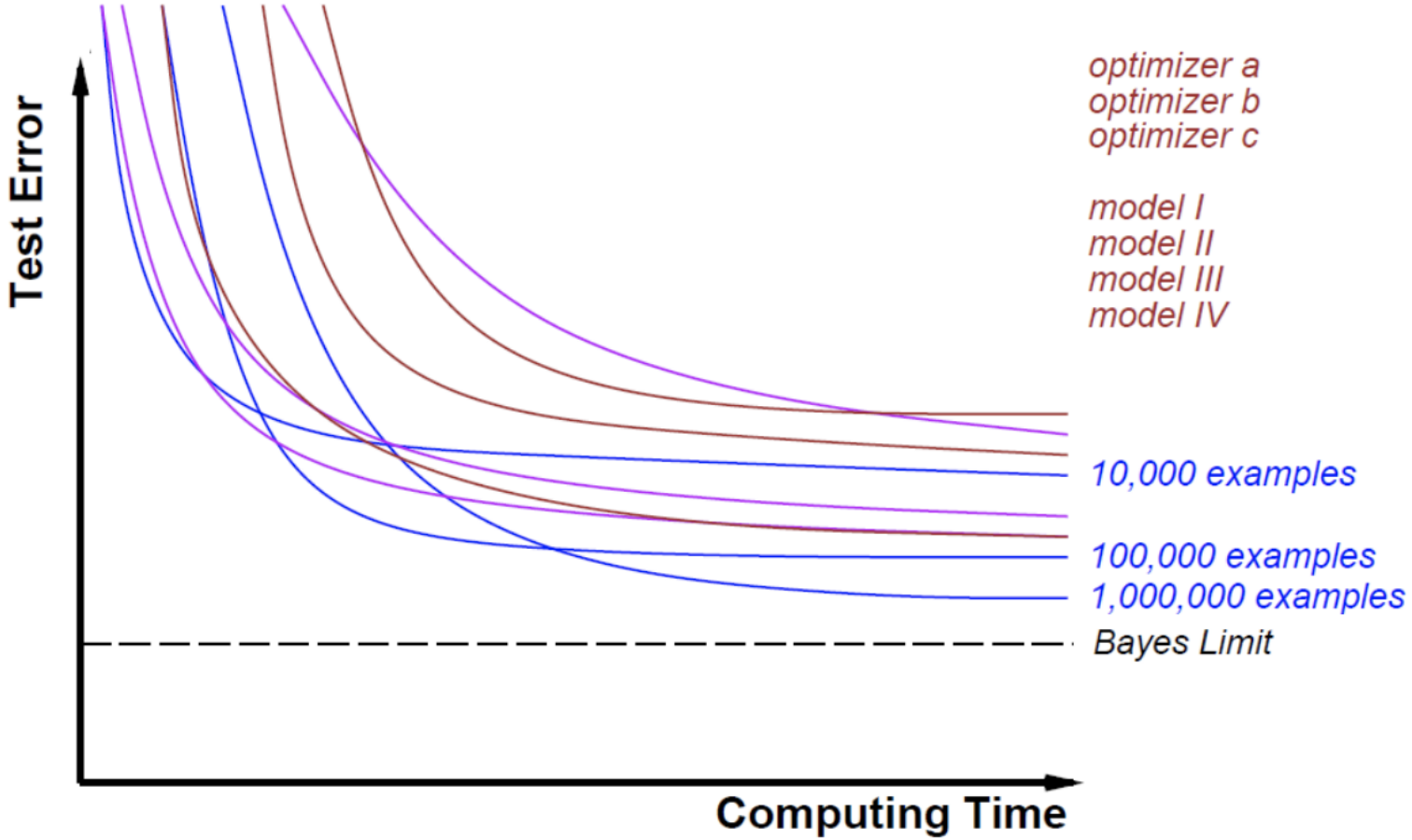


test error versus training time



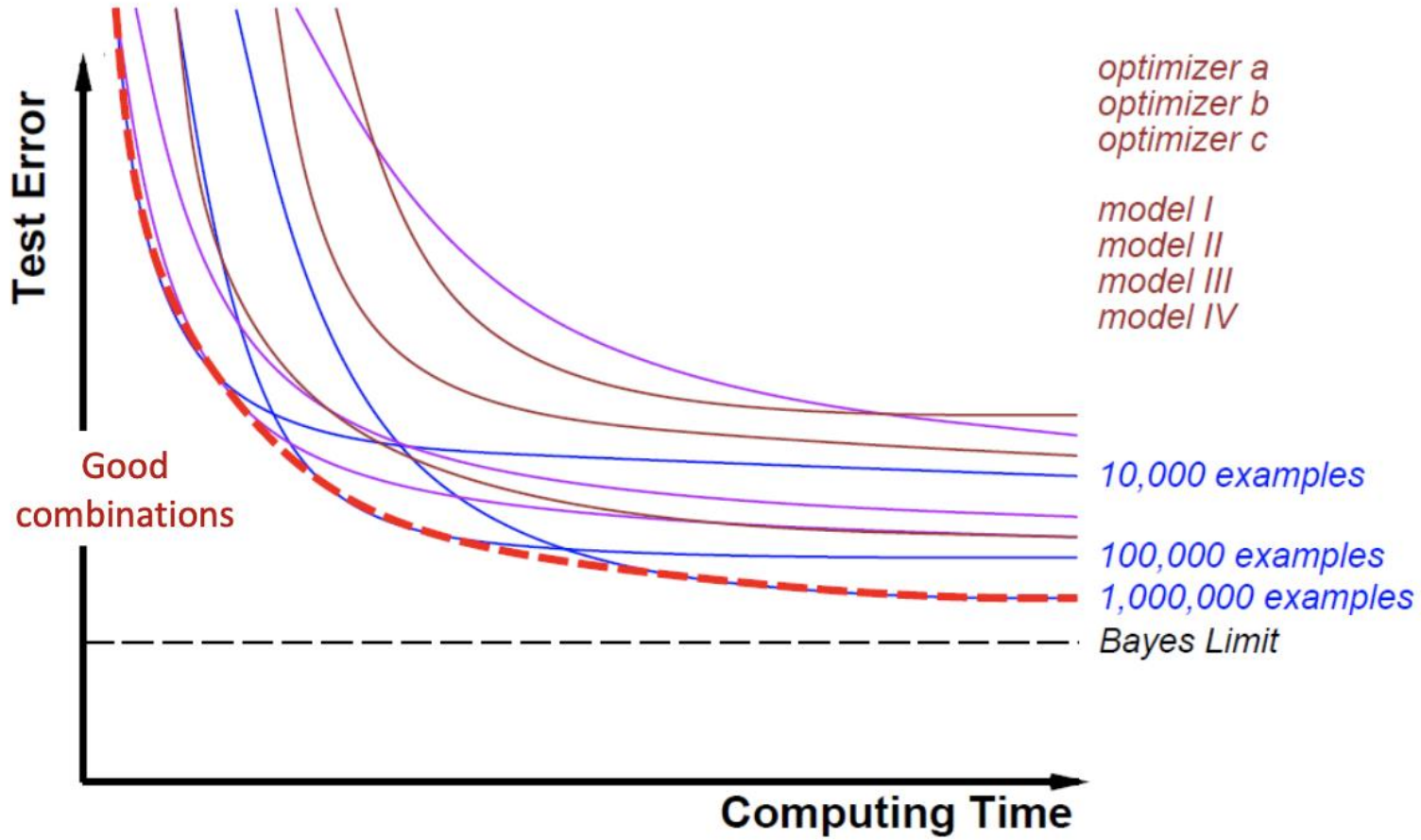
- Vary the number of examples

test error versus training time



- Vary the number of examples, the model, the algorithm

test error versus training time



- Optimal combination depends on training time budget.

the tradeoffs of large-scale learning

Small-scale learning ≠ large-scale learning

- Large-scale learning involves more complex tradeoffs
  that depends on the properties of the optimization algorithm.

Good optimization algorithm ≠ good learning algorithm

- Mediocre optimization algorithms (e.g., SGD)
  often outperform sophisticated optimization algorithms
  on large-scale learning problems.

provided that the code is correct (which is harder than it seems.)

Focusing on the data and the task

the value of big data in machine learning

Accuracy improvements are subject to diminishing returns.
Breadth improvements are not subject to diminishing returns.

"How accurately do we recognize an object category?"
vs. "How many categories do we recognize well enough?"

Should we optimize a different criterion?

How does this helps if average accuracy is what we care about ?

same distribution?

Data collection in traditional machine learning

- Training data collection for real-life machine learning is difficult.
  The data distribution must reflect the operational conditions.
- The i.i.d. assumption is not automatically satisfied.
  It happens through manual data curation.

Data collection in big data machine learning

- Big data exists because data collection is automated.
- No manual curation to enforce the identical distribution assumption.
- The output of the machine learning system frequently
  impacts the distribution of future training data.

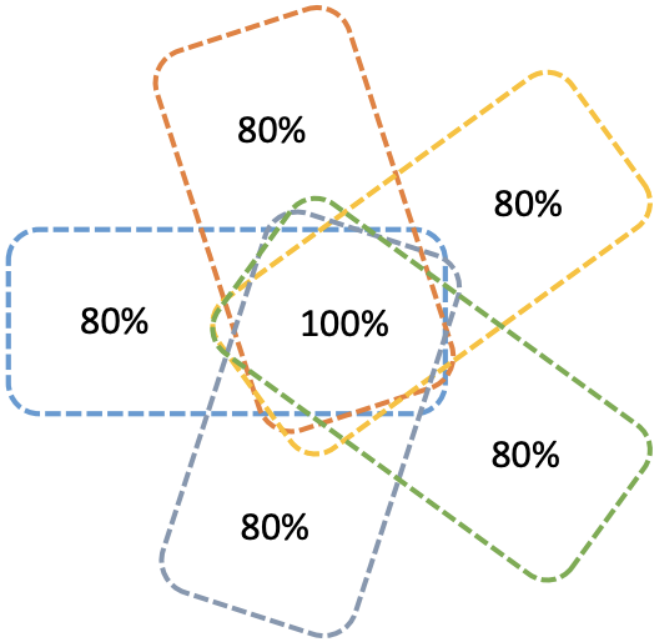
Search queries

- Lots of queries entered by lots of users.
- The search engines must satisfy the users, not the queries.
- The satisfaction of a user is not proportional to its satisfied queries.

independent examples?

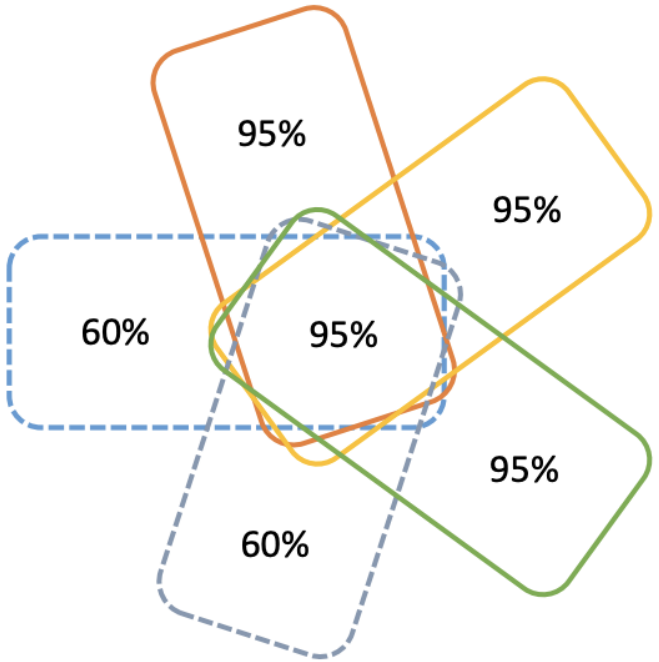
Assume that a user is not satisfied below 95% correct answers.

Minimize average error



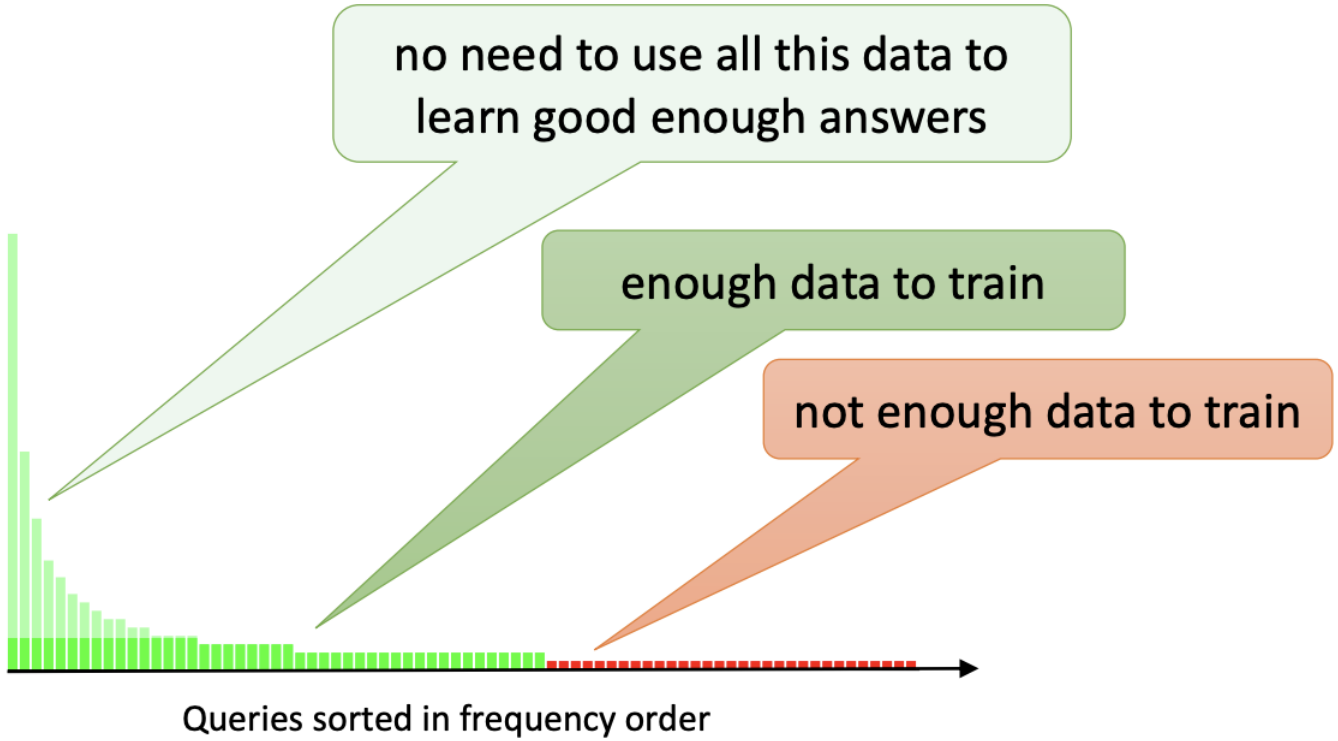
Average error: 10%
Users satisfied: 0

Maximize queries answered at 95% level



Average error: 12%
Users satisfied: 3

scalability opportunities



- No need to consider all examples of already known queries.
- Best is to focus on queries near the boundary of the known area.
- Curriculum learning and active learning come naturally in this context.
- Scalability gains across the board.

auxiliary tasks

The price of labels

Interesting task $\iff$ Scarce labeled examples.
Uninteresting task $\iff$ Abundant labeled examples.

Auxilliary tasks

– **"In the vicinity of an interesting task** (with expensive labels,)
there often are less interesting tasks (with cheap labels,)
that we can put to good use."

– Unsupervised learning is just one of them (with trivial labels.)

– **Deep-learning, semi-supervised learning, and transfer learning are three facets of the same thing.**

e.g. (Weston et al., 2008)

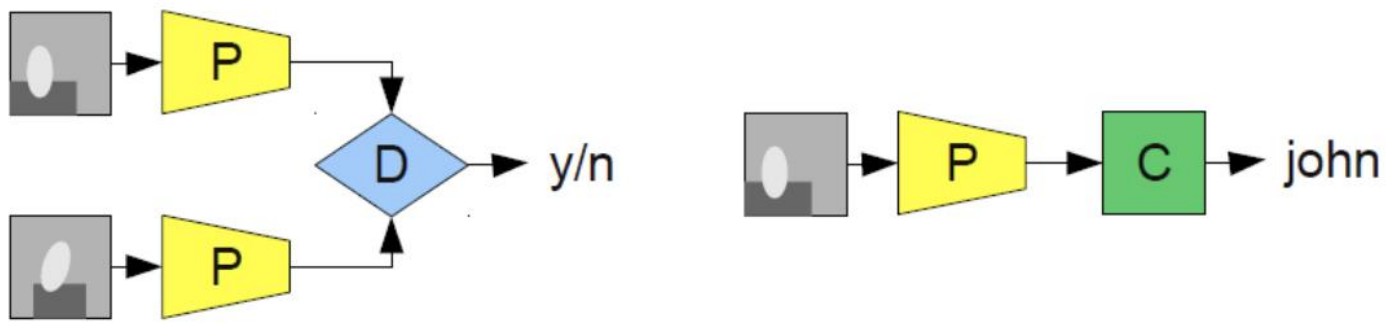
example – face recognition

Interesting problem

– Recognizing the faces of one million persons.
– How many labeled images per person will we get?

Related but less interesting problem

– Are two face images representing the same person?
– Abundant (but noisy) examples:
  ◇ Two faces in the same image are likely to be different persons.
  ◇ Faces in successive frames are likely to be the same person.



(Matt Miller, NECLA, 2006)

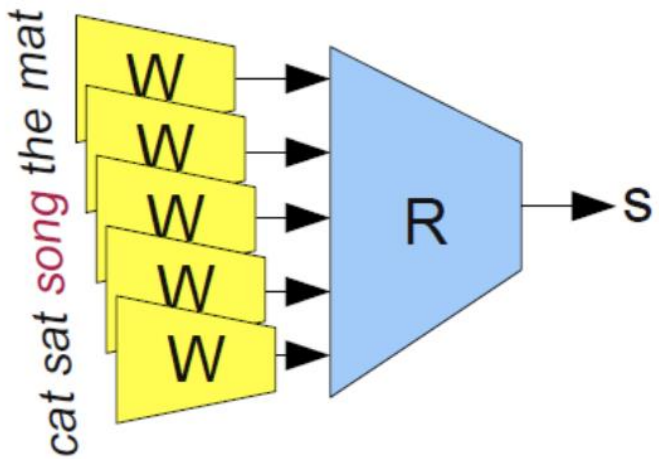
example – natural language tagging

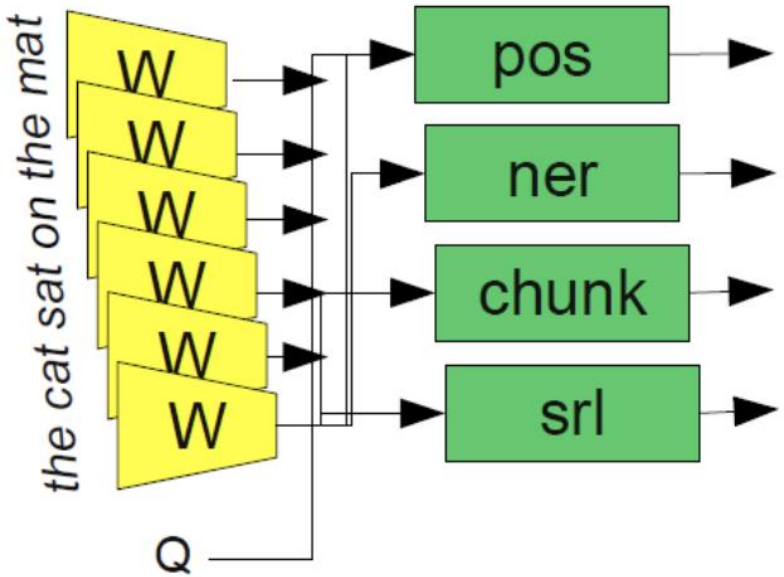
Interesting problems

– Standard NLP tagging tasks.

Related but less interesting problem

– Positive examples are legal sentence segments.
– Negative examples are created by substituting the central word.
– Ranking loss.





(Collobert et al., 2008-2011)

revisiting Vapnik's razor

" When solving a (learning) problem of interest, do not solve a more complex problem as an intermediate step. "

Rationale: how complex a model can we afford with our data?

However, solving a more complex task and transferring features often allows us to leverage more data of a different nature.

- Lots of implications.

"From machine learning to machine reasoning", L.B., 2011.

Summary

Large-scale machine learning is not about peta-bytes of data

It is about making complex tradeoffs due to time constraints (model complexity, number of examples, optimization algorithm, etc.)

Start with a known optimizer (e.g. Adam)

Focus on the data, the task and the metrics

Large Scale ML Frameworks

Spark MLlib

- ML library built on top of Spark
  - Scala/Java/python
  - Distributed
  - Breeze for linear algebra / numerical optimization in Scala
- Content
  - ML Algorithms: classification, regression, clustering, collaborative filtering, frequent pattern mining
  - Feature extraction and transformation
  - Pipelines
  - Persistence: load/store models and pipelines

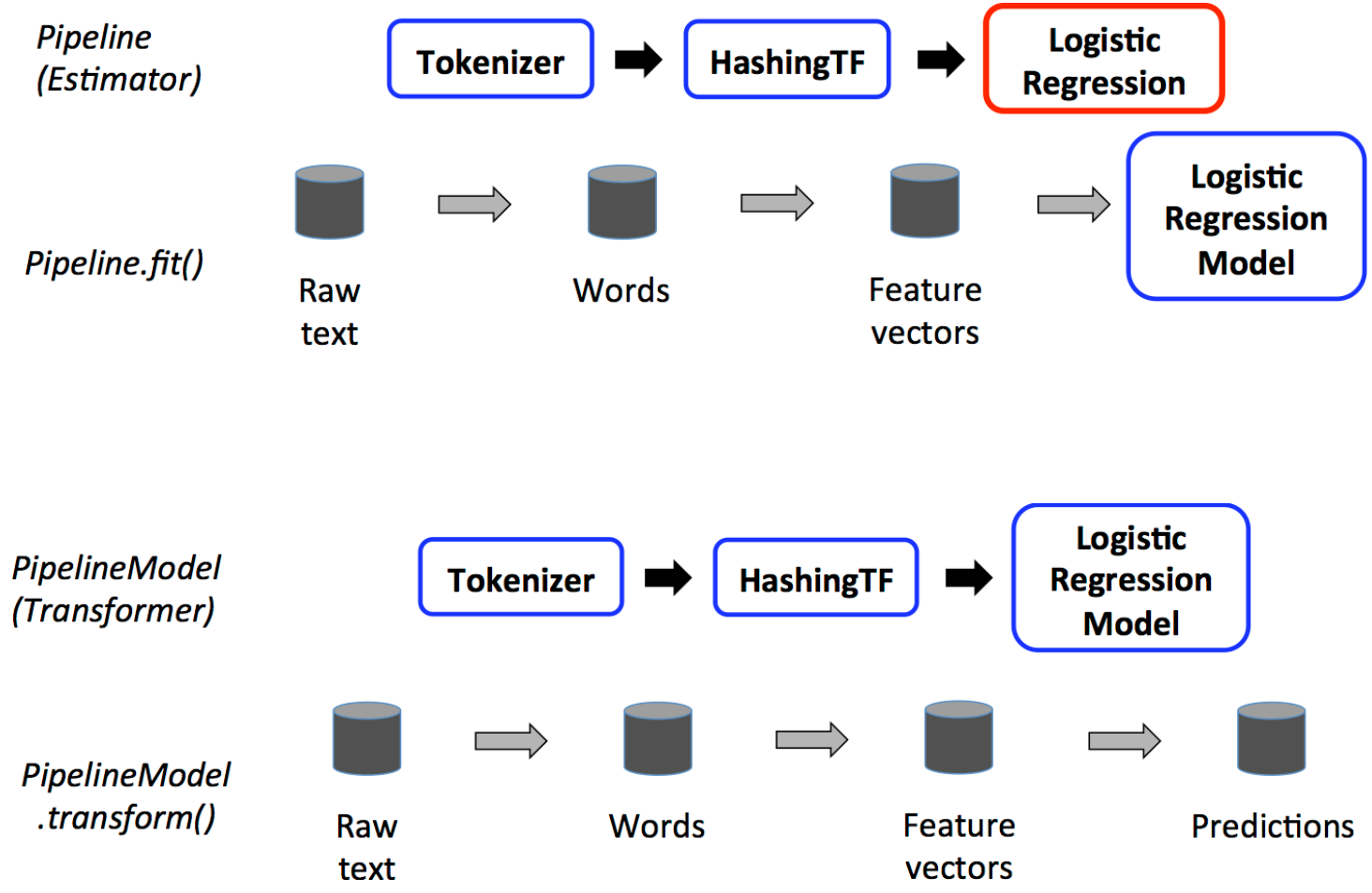
https://spark.apache.org/docs/latest/ml-guide.html

Spark MLLib: Components

- Dataframe
- Transformer
  - Transform: Dataframe -> Dataframe
  - eg: ML model features_df -> prediction_df
- Estimator
  - fit: Dataframe -> Transformer
  - eg: Learning algorithm
- Pipeline
  - A chain of transformers & estimators

Spark MLLib: Pipeline

- Split each document's text into words
- Convert each document's words into a numerical feature vector
- Learn a prediction model using the feature vectors and labels



Spark MLLib: Train pipeline

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

# Prepare training documents from a list of (id, text, label) tuples.
training = spark.createDataFrame([
    (0, "a b c d e spark", 1.0),
    (1, "b d", 0.0),
    (2, "spark f g h", 1.0),
    (3, "hadoop mapreduce", 0.0)
], ["id", "text", "label"])

# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(training)
```

Spark MLLib: Predict with pipeline

```
# Prepare test documents, which are unlabeled (id, text) tuples.
test = spark.createDataFrame([
    (4, "spark i j k"),
    (5, "l m n"),
    (6, "spark hadoop spark"),
    (7, "apache hadoop")
], ["id", "text"])

# Make predictions on test documents and print columns of interest.
prediction = model.transform(test)
selected = prediction.select("id", "text", "probability", "prediction")
```

Spark MLlib: Feature transformations

- Bucketizer

```
from pyspark.ml.feature import Bucketizer

splits = [-float("inf"), -0.5, 0.0, 0.5, float("inf")]

data = [(-999.9,), (-0.5,), (-0.3,), (0.0,), (0.2,), (999.9,)]
dataFrame = spark.createDataFrame(data, ["features"])

bucketizer = Bucketizer(splits=splits, inputCol="features", outputCol="bucketedFeatures")

# Transform original data into its bucket index.
bucketedData = bucketizer.transform(dataFrame)
```

- QuantileDiscretizer

```
from pyspark.ml.feature import QuantileDiscretizer

data = [(0, 18.0), (1, 19.0), (2, 8.0), (3, 5.0), (4, 2.2)]
df = spark.createDataFrame(data, ["id", "hour"])

discretizer = QuantileDiscretizer(numBuckets=3, inputCol="hour", outputCol="result")

result = discretizer.fit(df).transform(df)
```

Spark MLlib: Feature transformations

- TF-IDF
- Word2Vec
- CountVectorizer
- FeatureHasher
- Tokenizer
- StopWordsRemover
- n-gram
- Binarizer
- PCA
- PolynomialExpansion
- Discrete Cosine Transform (DCT)
- StringIndexer
- IndexToString
- OneHotEncoderEstimator
- VectorIndexer
- Interaction
- Normalizer
- StandardScaler
- MinMaxScaler
- MaxAbsScaler
- Bucketizer
- ElementwiseProduct
- SQLTransformer
- VectorAssembler
- VectorSizeHint
- QuantileDiscretizer
- Imputer

Spark MLlib: Algorithms

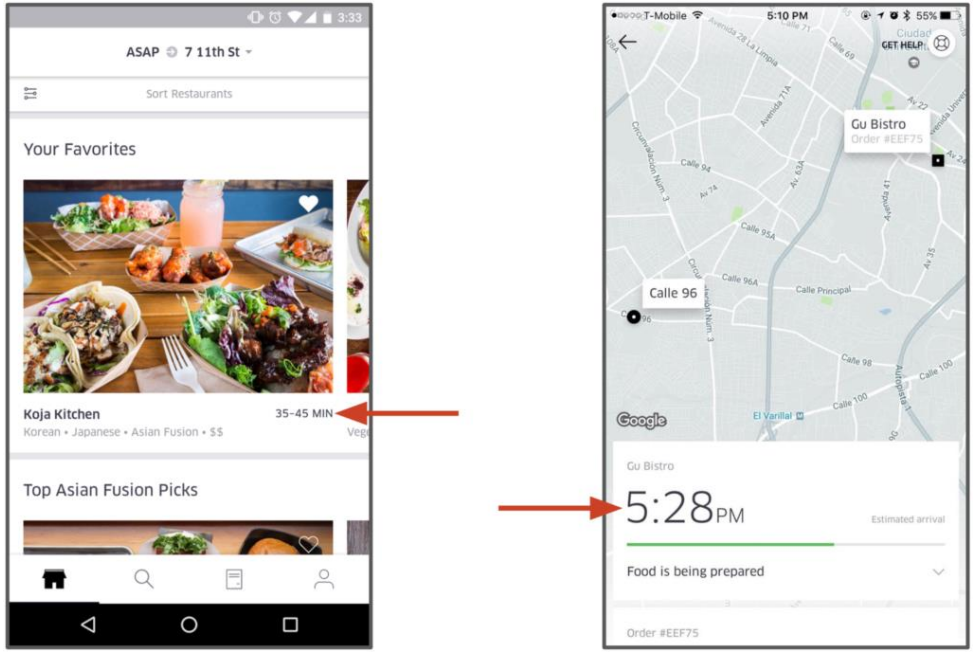
- Classification and regression
  - Generalized Linear Model
  - Decision tree
  - Random forest
  - Gradient-boosted decision tree
- Clustering
  - K-means
  - Latent Dirichlet allocation (LDA)
  - Gaussian Mixture Model (GMM)
- Collaborative filtering
- Frequent pattern mining
  - FP-Growth
  - PrefixSpan

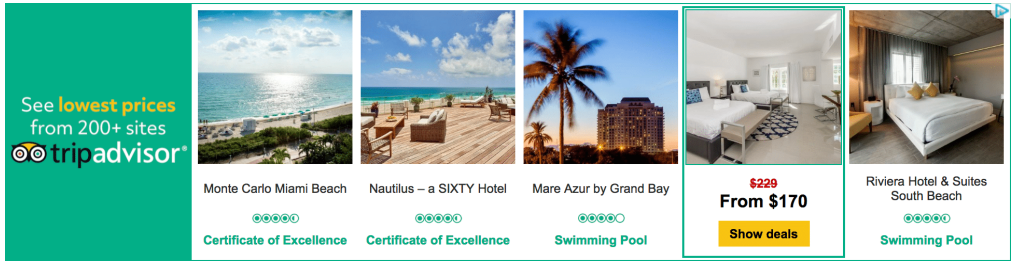
```
from pyspark.ml.classification import LogisticRegression

# Load training data
training = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Fit the model
lrModel = lr.fit(training)

# Print the coefficients and intercept for logistic regression
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))
```

Spark MLlib: Use cases in production

Uber

https://eng.uber.com/michelangelo-machine-learning-platform/



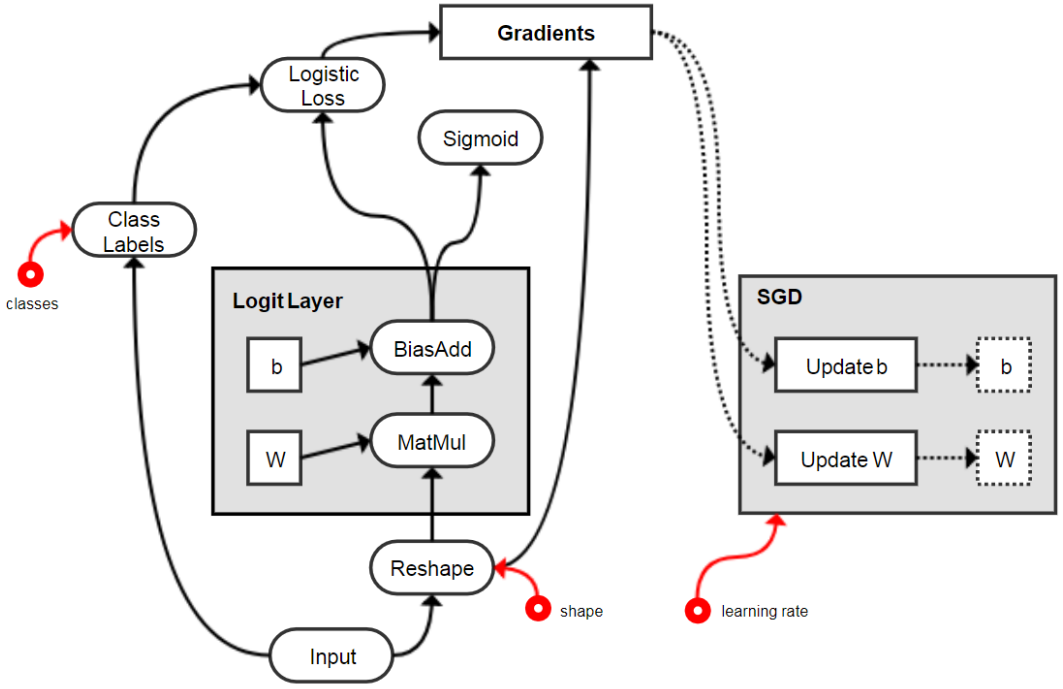


criteo

http://labs.criteo.com/2017/01/large-scale-machine-learning-criteo/

Tensorflow

- Numerical computation using data flow graphs
- Autodiff
- Parallel, distributed, accelerator support



https://www.tensorflow.org/

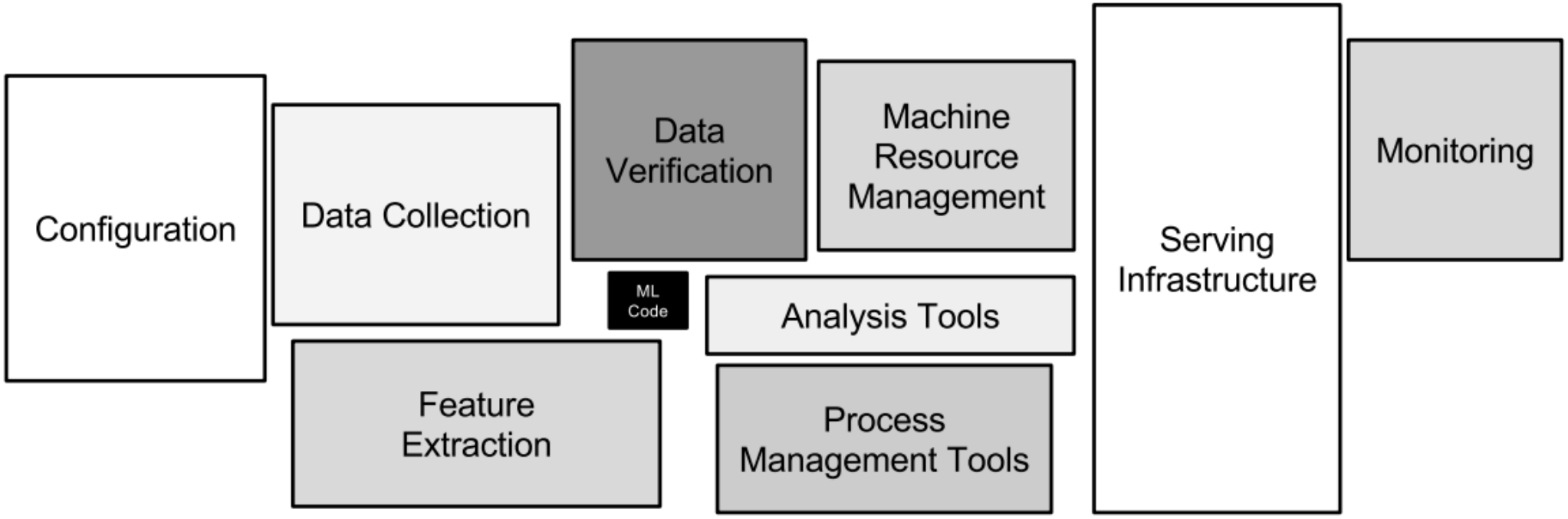
Tensorflow

- Python API, implemented in C++
- Keras high level api
- Graph low level api

```python
from tensorflow.keras import layers

model = tf.keras.Sequential()
# Adds a densely-connected layer with 64 units to the model:
model.add(layers.Dense(64, activation='relu'))
# Add another:
model.add(layers.Dense(64, activation='relu'))
# Add an output layer with 10 output units:
model.add(layers.Dense(10))
```

```python
def dense(x, W, b):
  return tf.nn.sigmoid(tf.matmul(x, W) + b)

@tf.function
def multilayer_perceptron(x, w0, b0, w1, b1, w2, b2 ...):
  x = dense(x, w0, b0)
  x = dense(x, w1, b1)
  x = dense(x, w2, b2)
  ...
```

ML Models are not everything



https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

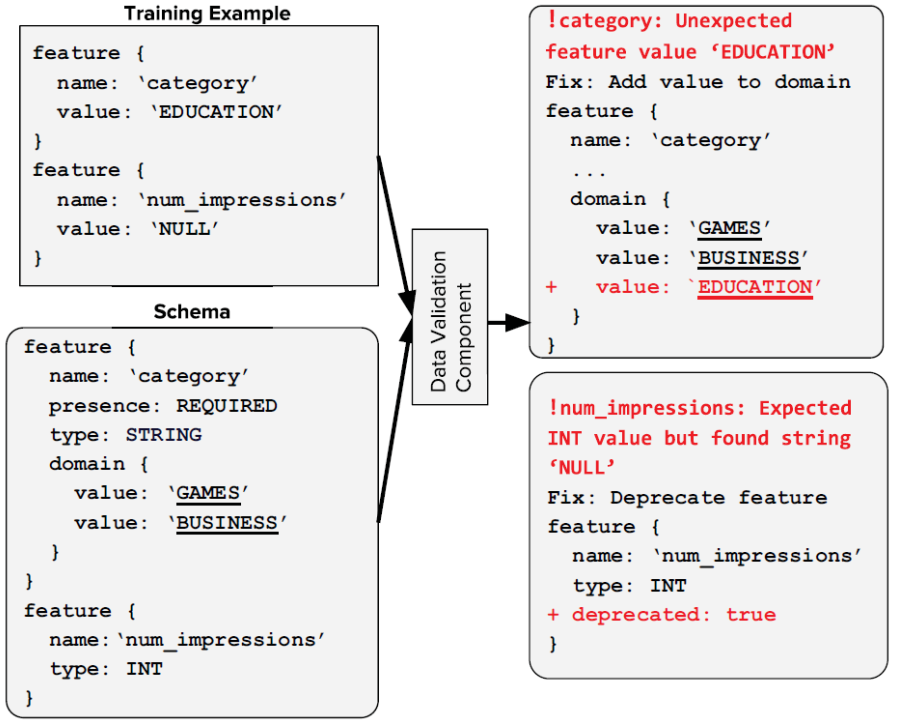
TensorFlow Extended

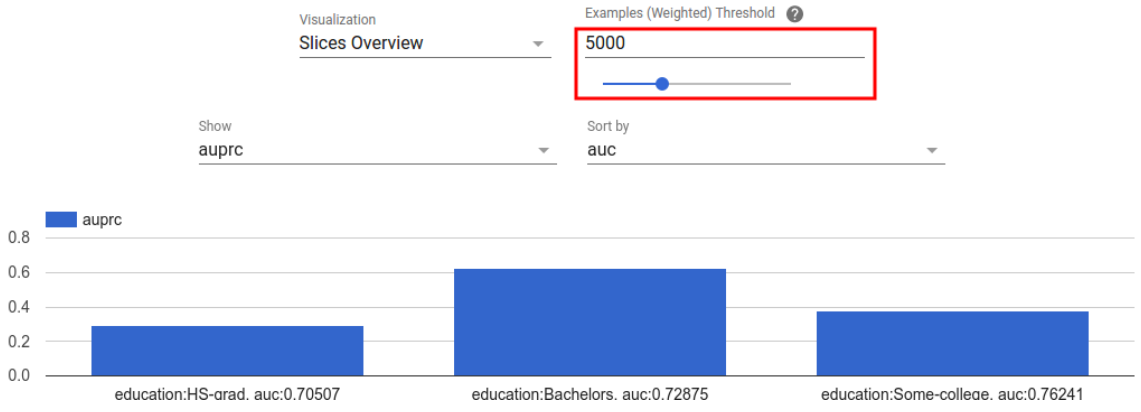
Data Validation

Model Analysis

Transform

Serving

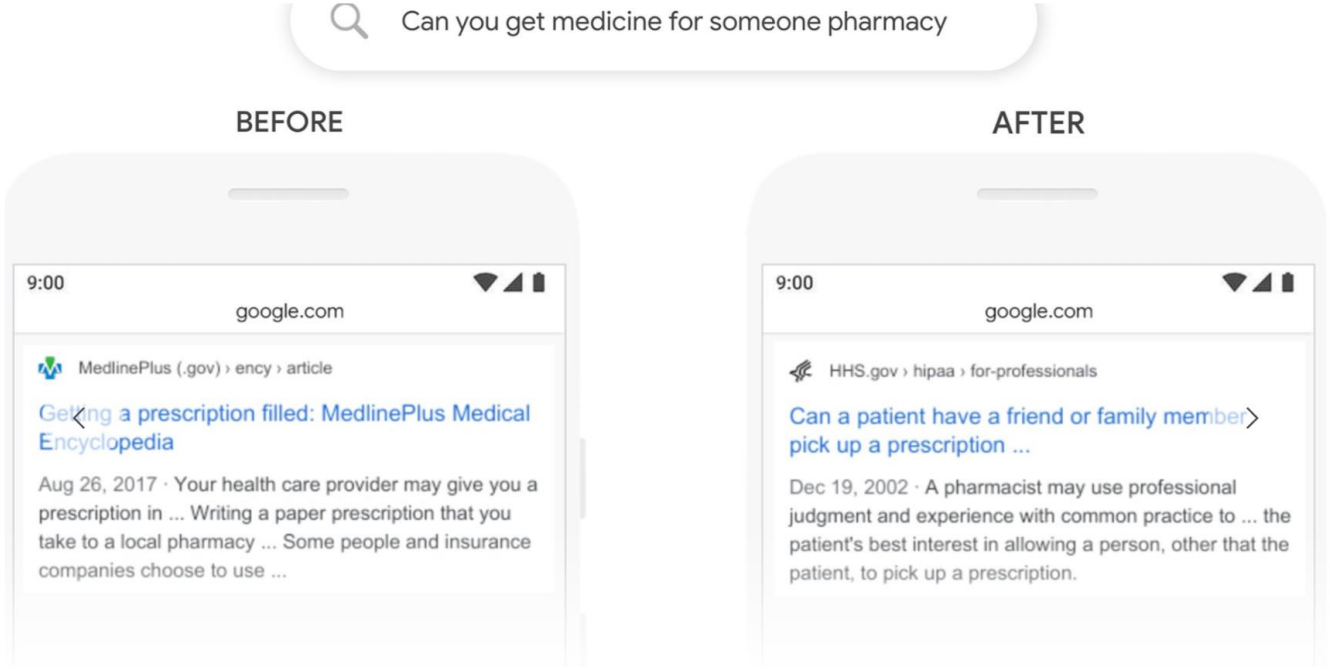




https://www.kdd.org/kdd2017/papers/view/tfx-a-tensorflow-based-production-scale-machine-learning-platform

https://www.tensorflow.org/tfx

Tensorflow: Use cases in production

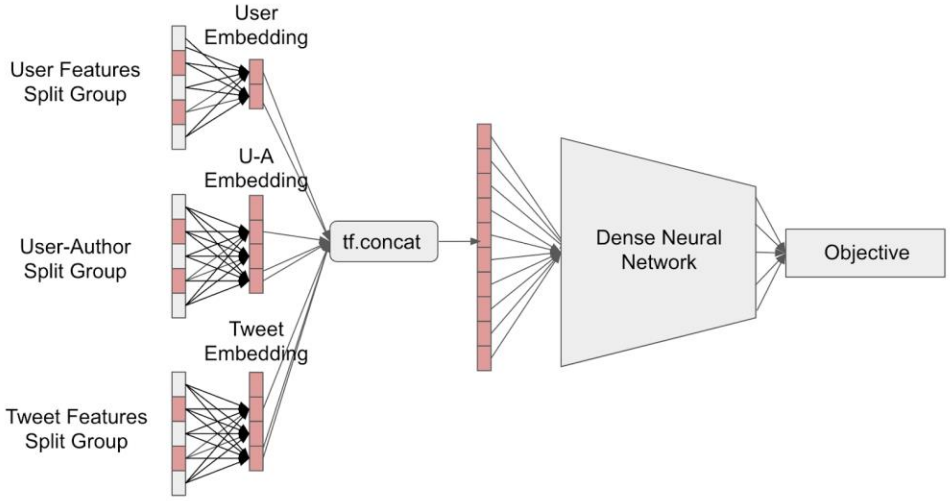


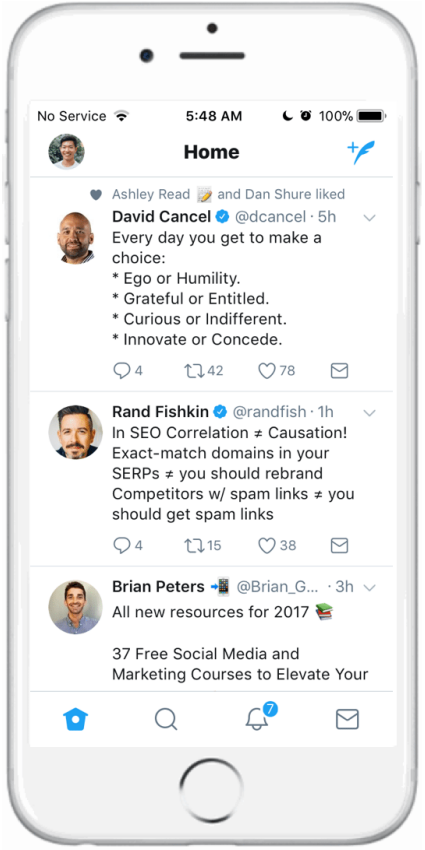
Google

"BERT will help Search better understand one in 10 searches in the U.S. in English"

https://blog.google/products/search/search-language-understanding-bert/

Tensorflow: Use cases in production





https://blog.tensorflow.org/2019/03/ranking-tweets-with-tensorflow.html

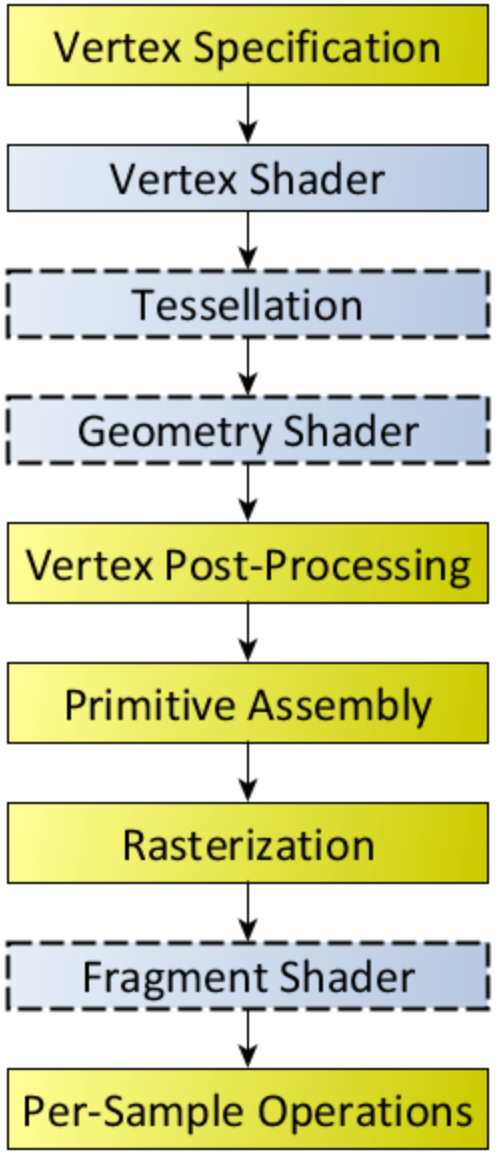
Hardware

GPU History

- 2001: DirectX / OpenGL shaders
  rendering pipeline becomes programmable
- 2007: Nvidia releases CUDA
- 2010: Most powerful computer in the world uses GPU

  https://www.top500.org/lists/2010/11/
- 2019: 40% of the compute power of the top500
  comes from GPU

  https://blogs.nvidia.com/blog/2019/11/19/record-gpu-accelerated-supercomputers-top500/



https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

CPU vs GPU: Specs

Intel® Xeon® Platinum 8280M Processor

# of Cores	28
# of Threads	56
Processor Base Frequency	2.70 GHz
Max Turbo Frequency	4.00 GHz
Cache	38.5 MB
TDP	205 W
Max Memory Size	2 TB
Memory Types	DDR4-2933
Maximum Memory Speed	2933 MHz
Memory Bandwidth	140 GB/s
Double-Precision Performance	2.42 TFLOPS (@ base freq)
Int8 Performance	19.3 TOPS

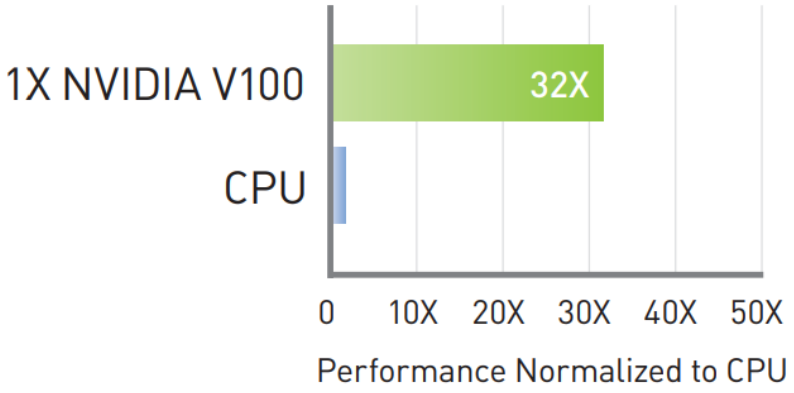
	V100 PCIe	V100 SXM2	V100S PCIe
GPU Architecture	NVIDIA Volta		
NVIDIA Tensor Cores	640		
NVIDIA CUDA® Cores	5,120		
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS	8.2 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS	16.4 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS	130 TFLOPS
GPU Memory	32 GB /16 GB HBM2		32 GB HBM2
Memory Bandwidth	900 GB/sec		1134 GB/sec
ECC	Yes		
Interconnect Bandwidth	32 GB/sec	300 GB/sec	32 GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink™	PCIe Gen3
Form Factor	PCIe Full Height/Length	SXM2	PCIe Full Height/Length
Max Power Comsumption	250 W	300 W	250 W
Thermal Solution	Passive		
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC®		

https://ark.intel.com/content/www/us/en/ark/products/192473/intel-xeon-platinum-8280m-processor-38-5m-cache-2-70-ghz.html

https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf

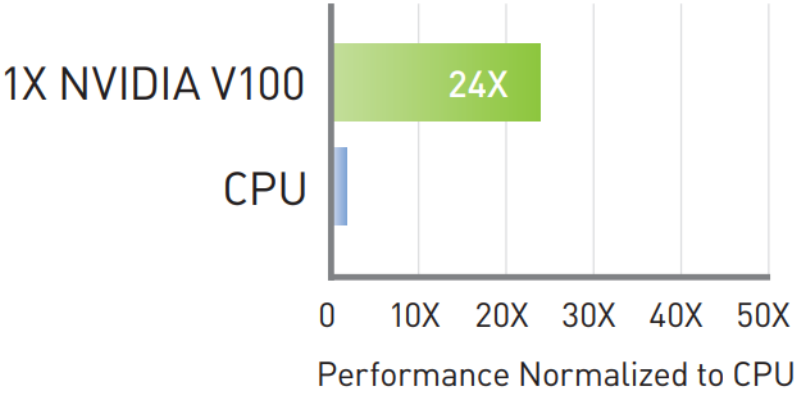
CPU vs GPU: Perf

32X Faster Training Throughput than a CPU[1]



ResNet-50 training, dataset: ImageNet2012, BS=256
| NVIDIA V100 comparison: NVIDIA DGX-2™ server, 1x V100 SXM3-32GB, MXNet 1.5.1, container=19.11-py3, mixed precision, throughput: 1,525 images/sec
| Intel comparison: Supermicro SYS-1029GQ-TRT, 1 socket Intel Gold 6240@2GHz/3.9Hz Turbo, Tensorflow 0.18, FP32 (only precision available), throughput: 48 images/sec

24X Higher Inference Throughput than a CPU Server[2]

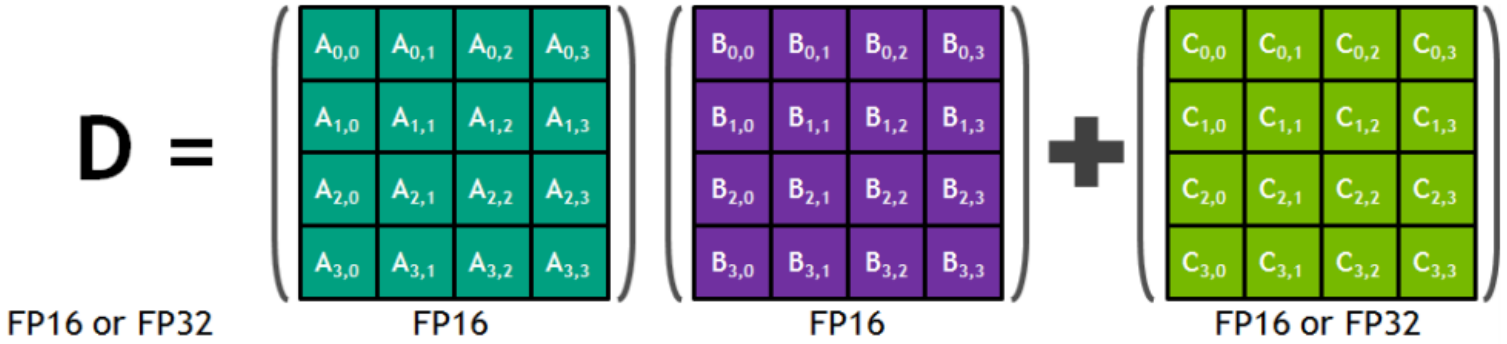


BERT Base fine-tuning inference, dataset: SQuADv1.1, BS=1, sequence length=128
| NVIDIA V100 comparison: Supermicro SYS4029GP-TRT, 1x V100-PCIE-16GB, pre-release container, mixed precision, NVIDIA TensorRT™ 6.0, throughput: 557 sentences/sec
| Intel comparison: 1 socket Intel Gold 6240@2.6GHz/3.9Hz Turbo, FP32 (only precision available), OpenVINO MKL-DNN v0.18, throughput: 23.5 sentences/sec

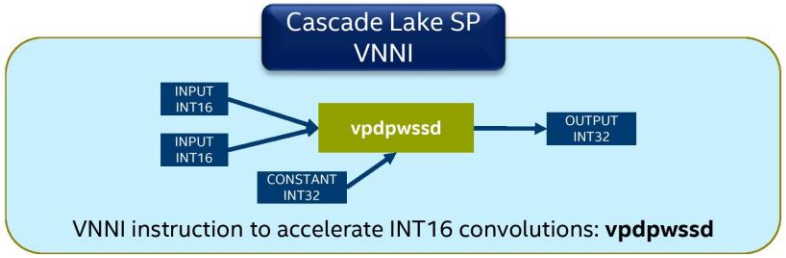
https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf

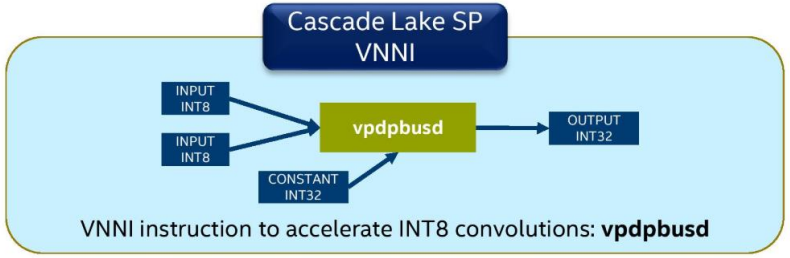
CPU & GPU: ML Specialized Instructions

Nvidia Tensor Core



Intel VNNI

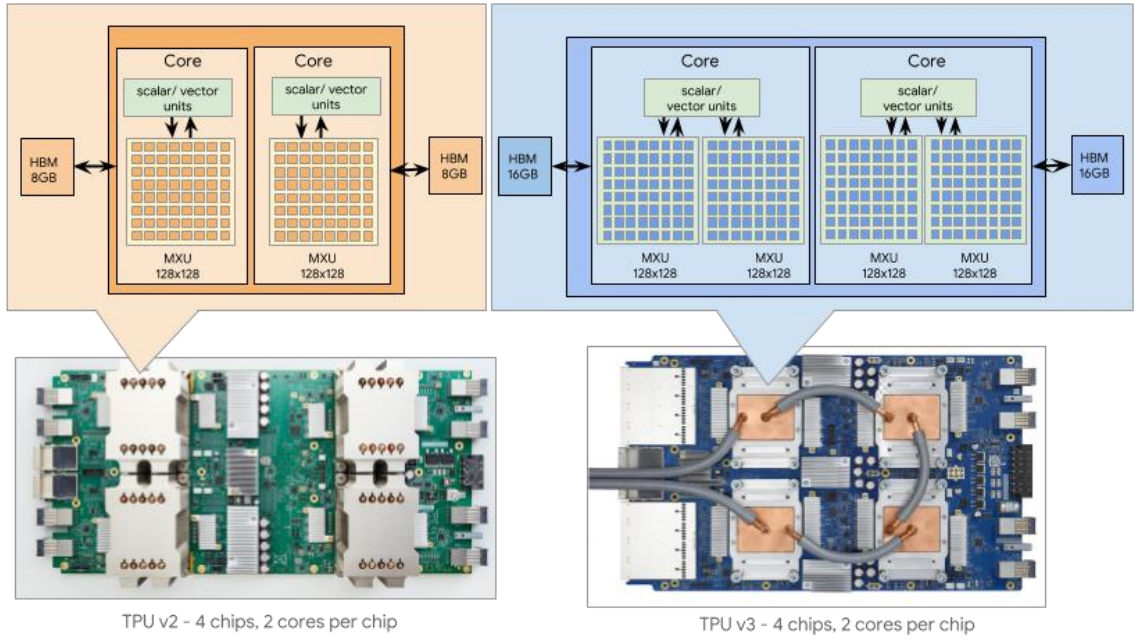




https://software.intel.com/en-us/articles/lower-numerical-precision-deep-learning-inference-and-training

https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

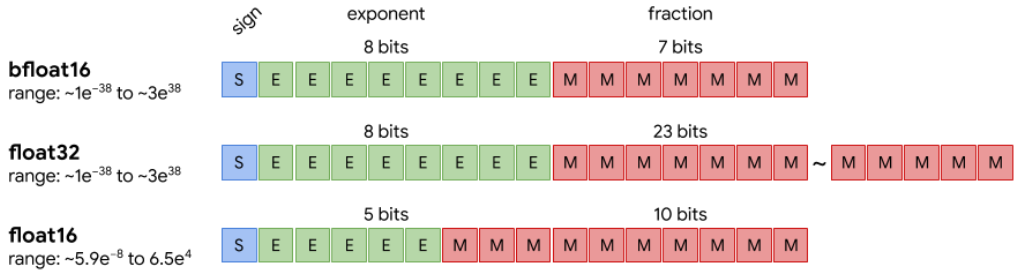
Even more specialized: Google TPU



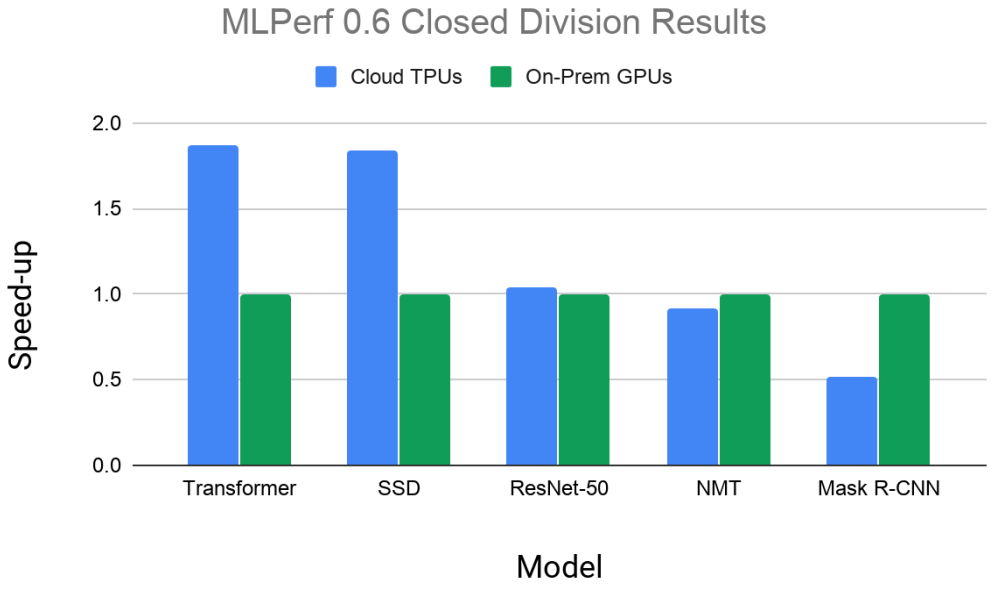
TPU v2 - 4 chips, 2 cores per chip

TPU v3 - 4 chips, 2 cores per chip

MXU: 22.5 TOPS (bfloat16)



https://cloud.google.com/tpu/docs/system-architecture



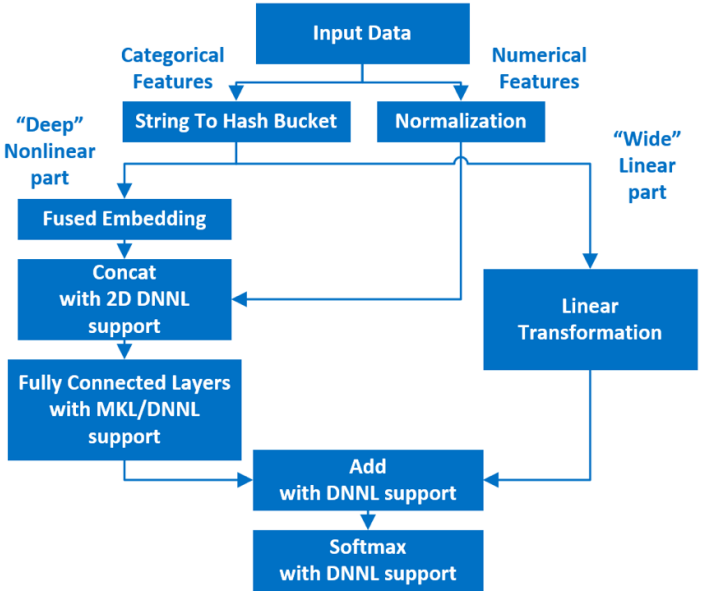
Google Cloud TPU v3 Pod speed-ups over the largest-scale on-premise NVIDIA DGX-2h clusters entered in the MLPerf 0.6 Closed Division. The Cloud TPU Pod submissions use 1024, 1024, 1024, 512, and 128 chips respectively; the NVIDIA DGX-2h clusters use 480, 240, 1536, 256, and 192 chips respectively. [1,2]

https://cloud.google.com/blog/products/ai-machine-learning/cloud-tpu-pods-break-ai-training-records

High Performance Libraries

- BLAS: linear algebra
  - Vector and matrix operations
  - MKL optimized for Intel CPUs
  - cuBLAS optimized for Nvidia GPUs

- Specialized libraries for deep learning
  - Fully connected layers, convolutions, RNN cells
  - Support for specialized HW instructions (tensor core, vnni)
  - mklDNN / cuDNN
  - Integrated into popular DL frameworks



All of this great, but I don’t have a GPU, TPUs or a cluster

All I have is this shitty laptop

Help my data does not fit in RAM !

- One of the most common problems encountered
- And most of the times it starts with something like

```
df = pd.read_csv("my_big_file.csv")
```

Pandas optimization

- Pandas is quite memory inefficient
- One issue is that at column type inference time, it casts them to the largest possible type
- Pandas types
  - **int8 / uint8 :** consumes 1 byte of memory, range between -128/127 or 0/255
  - **bool** : consumes 1 bit, true or false
  - **float16 / int16 / uint16:** consumes 2 bytes of memory, range between -32768 and 32767 or 0/65535
  - **float32 / int32 / uint32 :** consumes 4 bytes of memory, range between -2147483648 and 2147483647
  - **float64 / int64 / uint64:** consumes 8 bytes of memory

Pandas optimization

- Categorical columns
  - If you have some categorical columns in your dataset (with strings inside for instance) they are stored as objects
  - If the number of possible categories is limited you can force pandas to use a virtual mapping table where all unique values are mapped via an integer instead of a pointer. This is done using the **category datatype**.

Pandas optimization

- So what can you do ?

- You can:
  - Inspect a representative number of lines of your dataset
  - Downcast or convert into categorical the appropriate columns
  - Use this new schema to load your whole dataset

Pandas optimization

```
df = pd.read_csv("my_big_file.csv", nlines=20)

df["float_col"] = pd.to_numeric(df["float_col"], downcast="float")
df["int_col"] = pd.to_numeric(df["int_col"], downcast="unsigned")
df["cat_col"] = df["cat_col"].astype("category")

dtypes = df.dtypes
colnames = dtypes.index
types = [i.name for i in dtypes.values]
column_types = dict(zip(colnames, types))

df_optimized = pd.read_csv("my_big_file.csv", dtype=column_types)
```

Pandas optimization

- Don’t forget to only load the subset of columns you are going to use

```
df_optimized = pd.read_csv("my_big_file.csv", dtype=column_types, usecols=["float_col","int_col","cat_col"])
```

Help my data still does not fit in RAM !

- Don’t panic
  - Maybe using a smaller version of the dataset would help

- Sampling !
  - Uniformly sampling data
  - Subsampling some majority classe(s)
    - We use this at Criteo

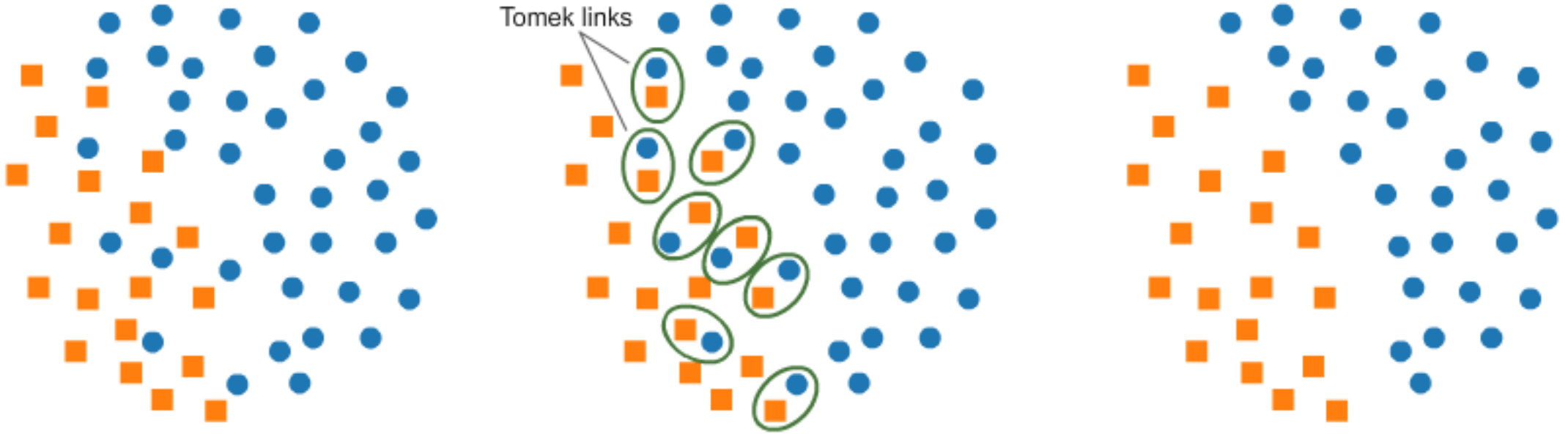
Using class imbalance to our favor

- Condensed nearest neighbours
  - Was originally developed to reduce the memory requirements for k-Nearest Neighbours predictors
  - Stream through the examples and only add an example if it cannot be correctly classified by the content of the added examples so far (using nearest neighbour methods)
  - This is a heuristic for adding informative examples and can be used only to subsample the majority class

Using class imbalance to our favor

- Tomek Links
  - Closest pairs of examples that have differing classes
  - Tomek Links are often misclassified examples found along the class boundary
  - Delete the example coming from the majority class

Using class imbalance to our favor



Help my data still does not fit in RAM !

- Actually it does not need to fit in RAM to build Machine Learning models
  - As long as it fits on your disk
  - Stream data from disk

Out of core algorithms

- The term *out-of-core* typically refers to processing data that is too large to fit into a computer's main memory.

- Replace random access to data in memory by sequential data access on disk (random access on disk is crazy expensive)

We need to shuffle the data first !

- How do you shuffle a file that does not fit in RAM ?
  - A solution: Tag each record with a random number. Pick random numbers from a very large set so that the probability that any two lines have the same random number is small. Then use external-memory sorting.
  - Some versions of the UNIX utility sort already have this

```
cat -n myfile.csv | sort --random-sort | cut -f 2-
```

We need to shuffle the data first !

- How do you shuffle a file that does not fit in RAM ?
  - A better solution: Shuffling is possible in linear time O(n). Sorting is a harder problem (in $O(n\ log\ n)$).
  - Can we shuffle in linear time without random access with variable-length records?

    - Create N temporary files, choose N large enough so that your entire set divided by N is likely to fit in RAM.
    - Assign each string to one temporary file at random.
    - Shuffle the temporary files in RAM.
    - Concatenate the temporary files.

Mini-batch Stochastic Gradient Descent

- Dataset $\{(x_i, y_i)_{i=1..N}\}$
  - Grouped into batches $\{(X_b, Y_b)_{b=1..B}\}$
- Differentiable predictor $f_w(X)$
- Cost function $J(f_w(X), Y)$

Mini-batch Stochastic Gradient Descent

- Choose initial vector of parameters w and learning rate α
- Repeat until some stopping criterion is met
  - For $b = 1\ to\ B$
    - $w := w - \alpha \nabla_w J(X_b, Y_b)$
- We only need one batch to fit into memory !

Large scale supervised ML in Scikit-learn

- **Estimators** that support mini-batch fitting (such as mini-batch optimization) have a **partial_fit** method

- Examples
  - SGDClassifier
  - Perceptron
  - MultinomialNB
  - …

Large scale supervised ML in Scikit-learn

```
model = SGDClassifier()

for X_b, y_b in batch_generator():
        model.partial_fit(X_b, y_b)
```

Large scale supervised ML in Scikit-learn

- SGDClassifier supports multiple losses (hinge, log-loss, …)

- Has a lot of nice features
  - In the case of sparse feature vectors, the intercept is updated with a smaller learning rate (multiplied by 0.01) to account for the fact that it is updated more frequently

  - Learning rate is lowered after each observed example with a schedule that adapts to the loss

We are not limited by RAM anymore !

Help I now have too many features !

- I have quite a few high cardinality categorical columns
  - Even building a vocabulary of possible values is a pain
- I want to use cross features and I have too many of them
- I want to limit the size of my model

The hashing trick

The hashing trick

- A **hash function** is any function that can be used to map data of arbitrary size to fixed-size values (most of the times to integers)

- In our case we will try to map categorical data to a a bounded value using hashing
  - Imagine all features are categorical (numerical features can be mapped to categorical features by bucketization)
  - Choose a fixed size for the feature vector, let's say we use 2^b (so we map our categorical features to a b-bit integer)
  - For each row of data and each feature, use the hash function to map the modality to the appropriate hash bucket

The hashing trick

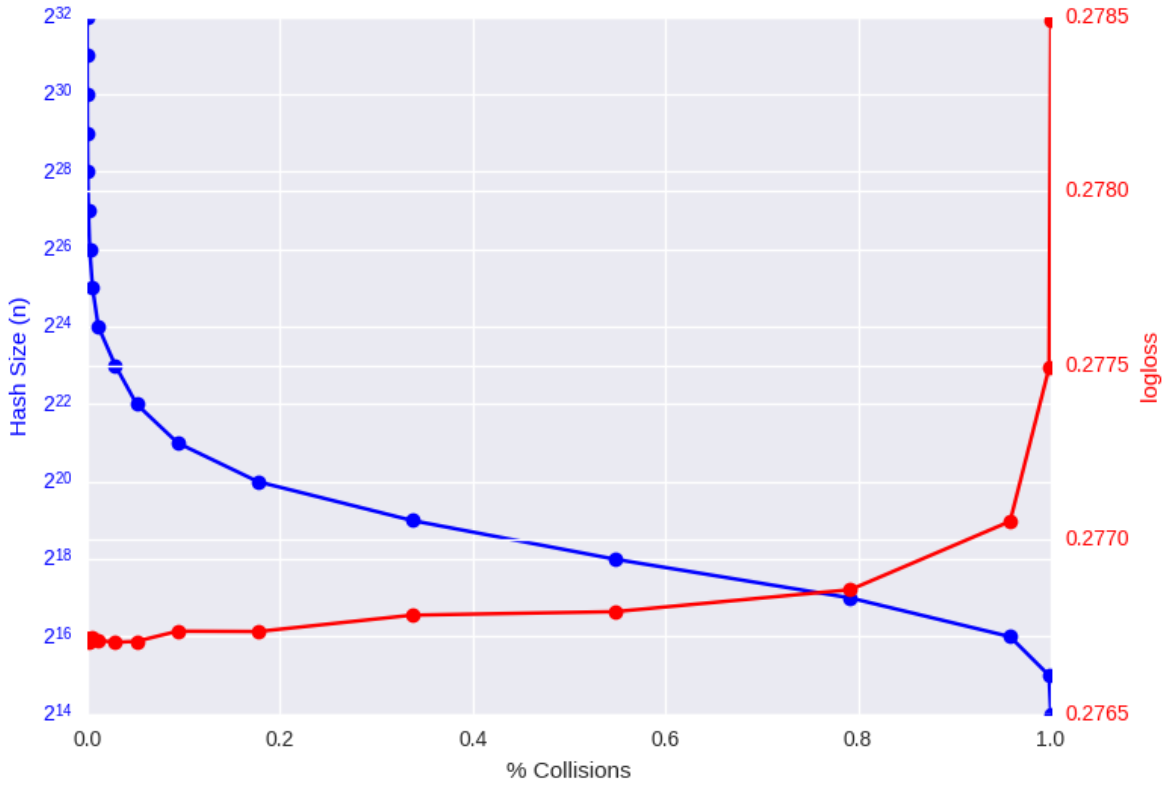
- Problem ?
  - Collisions !
  - What can we do ?
    - Ostrich algorithm: Ignore them …
    - Reserve hash buckets for important features

The hashing trick

- Collisions are actually not that bad, an experiment
  - Booking.com's internal dataset containing ~200k features and ~250M examples for a 4 classes classification problem
  - The Criteo Display Advertising Kaggle Challenge data set, which contains ~30M features and ~40M examples
  - The Avazu CTR Prediction Kaggle Challenge with about ~40M features and ~40M examples.

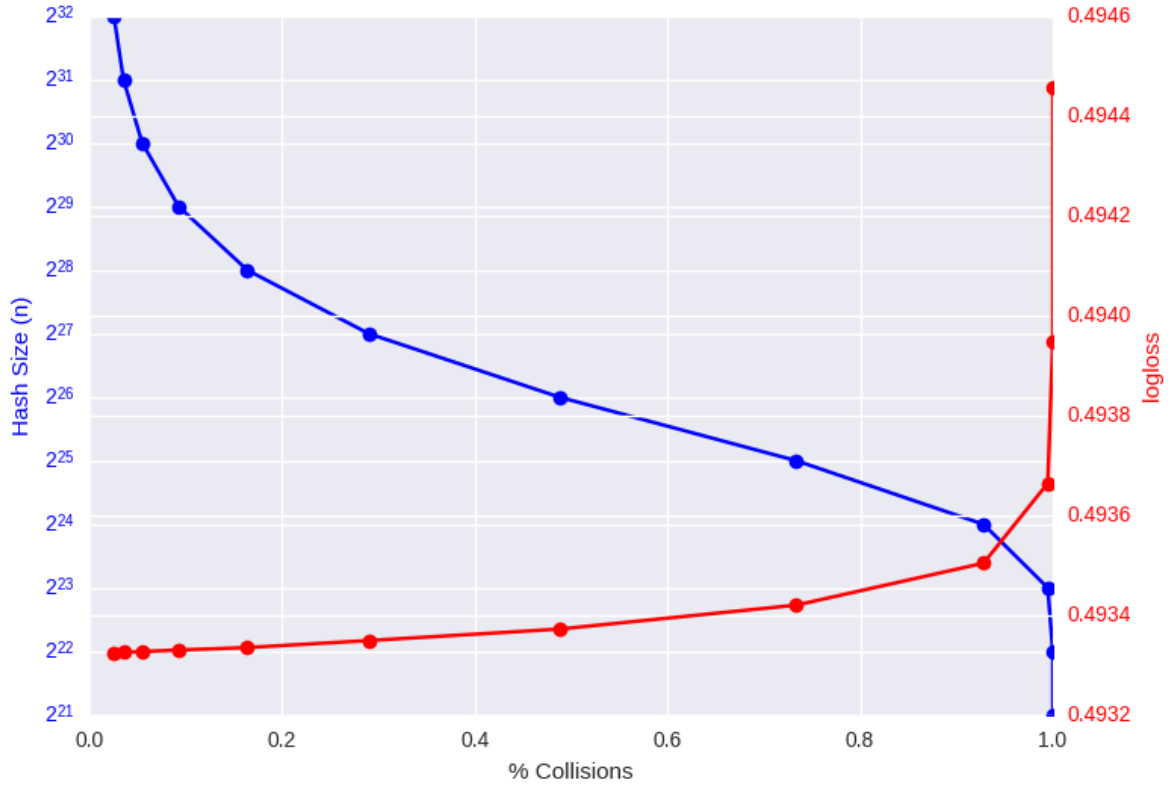
Source https://booking.ai/dont-be-tricked-by-the-hashing-trick-192a6aae3087

The hashing trick



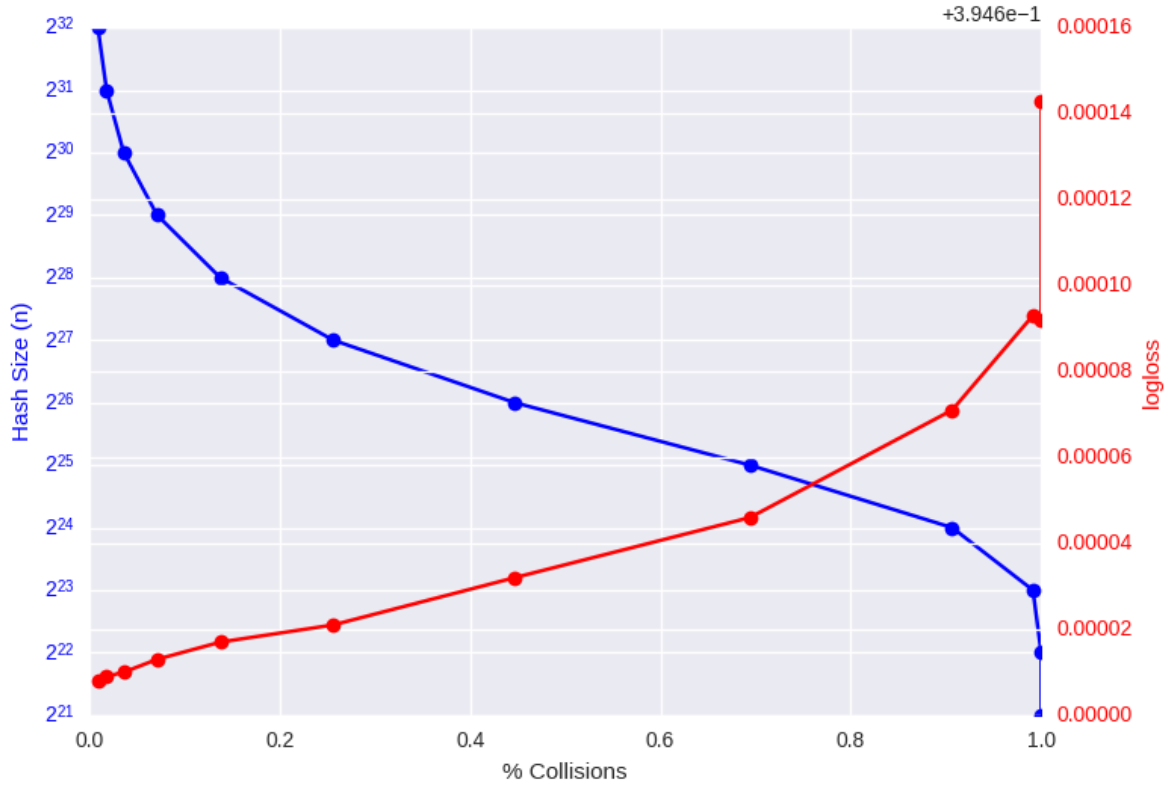
Booking dataset

The hashing trick



Criteo dataset

The hashing trick



Avazu dataset

The hashing trick

- Another problem
  - Potential loss of interpretability if there are collisions (weights will correspond to multiple modalities), but there are ways …

- Why not just use as big a coefficients table as possible?
  - That would certainly minimise collisions — but at the expense of memory, which defeats one of the purposes of the Hashing Trick: *to avoid the consumption of memory by the features*.

Ok memory is less of an issue now

- What is the new bottleneck ?
  - Disk IO
  - CPU

- In an ideal world a gradient step would be so fast that the only **bottleneck** left is **IO**, where training is as fast as reading data

Vowpal Wabbit

- State of the art in scalable, fast, efficient Machine Learning. VW is (by far) the most scalable public linear learner, and plausibly the most scalable anywhere (in a single machine setup).
- Usable from a (not so intuitive) command line
- Pure Online Learning (one example at a time)
- Uses feature hashing extensively
- Very weird input format

https://github.com/VowpalWabbit/vowpal_wabbit

Vowpal Wabbit: Input format

Label [Importance] [Base] [Tag] |Namespace Feature ... | Namespace Feature ... ... \n

- Namespace = String[:Float]
- Feature = String[:Float]
- If String is an integer, that index is used, otherwise a hash function computes an index. Float is the (optional) value of the feature
- Feature and Label are what you expect.
- Importance is multiplier on learning rate, default 1.
- Base is a baseline prediction, default 0.
- Tag is an identifier for an example, echoed on example output. Namespace is a mechanism for feature manipulation and grouping.

Vowpal Wabbit: Input format

-1 |i F0:58 F5:2143 F11:261 F12:1 F13:-1 F14:0 |c F1=management F2=married F3=tertiary F4=no F6=yes F8=unknown F9=5 F10=may F15=unknown

Vowpal Wabbit

- **Train**

```
vw train.vw -f model.vw --loss_function logistic
```

- **Test**

```
vw test.vw -t -i model.vw -p preds.txt --link logistic
```

Vowpal Wabbit

- Train

vw train.vw -f model.vw --loss_function logistic

  - **-f** output name of the model file
  - **--loss_function** loss to use (squared,logistic,hinge,quantile)

Vowpal Wabbit

- Test

vw test.vw -t -i model.vw -p preds.txt --link logistic

  - **-t** test mode (ignore labels)
  - **-i** model file to use
  - **-p** prediction output file
  - **--link logistic** apply the sigmoid function to the predictions

Vowpal Wabbit

- Some other useful options
  - **-c --passes N** This specifies to do N passes on the training set while learning the optimal weights. Note that the -c option specifying to use caching is necessary when doing multiple passes because from the second pass, VW is using pre-compiled information that it prepared/cached during the first pass.
  - **-b N** The -b option allows you to control the number of bits in the hashing namespace
  - **--interactions arg** arg is a list of letters, and each letter represents a namespace. Applying that option means that it will automatically create interactions between all features in the corresponding namespaces. For instance, in our example above, adding e.g. --interactions ic will create all the interactions pairs between features in the namespace i and c .

Vowpal Wabbit

Cool ! But my problem is highly non linear

- First off
  - Are you sure ? Did you try using quadratic interactions, low rank quadratic interactions (e.g. factorization machines)

- Yes I am sure !
  - **XGBoost**: Extreme Gradient Boosting

dmlc
XGBoost

Gradient boosting

- Boosting
  - Iteratively fit predictors on the data by taking into account the mistakes done by the previous predictor
- Gradient boosting
  - Iteratively fit predictors on the residual (i.e. prediction – true value) of the previous predictors and combine them using a learning rate
  - But why gradient ?

Gradient boosting

- Using mean squared error $\frac{1}{n}\sum_i (\widehat{y_i} - y_i)^2$
- For stage m
  - Find a new predictor $h(x)$ and add it to the predictor we have so far $F_{m-1}(x)$

$$F_m(x) = F_{m-1}(x) + h(x)$$

  - h needs to fit the residual $y - F_{m-1}(x)$
- But $y - F(x)$ is just the negative of the gradient (w.r.t. F(x)) of $\frac{1}{2}\big(y - F(x)\big)^2$
  - It's like we are doing gradient descent in functional space !
  - This is how we generalize to other losses

XGBoost

- Very efficient implementation of Gradient Boosting with a lot of trick to make everything efficient

- Supports out of core computation so your data does not need to fit into memory

XGBoost

- Optimization example
  - Data is converted to Compressed Sparse Columnar format
    - Very common sparse data format
    - Efficient processing of columns in a sequential manner
  - Values in the columns are sorted

Compressed Sparse Column (CSC)

41	0	0	44	0
0	22	0	0	25
0	0	33	14	0
11	0	43	35	0

ROW		ENTRY	ENTRY
0	Column 0 begins	41	0
3		11	2
1	Column 1 begins	22	3
2	Column 2 begins	33	5
3		43	8
0	Column 3 begins	44	
2		14	
3		35	
1	Column 4 begins	25	

Sorted Compressed Sparse Column (SCSC)

41	0	0	44	0
0	22	0	0	25
0	0	33	14	0
11	0	43	35	0

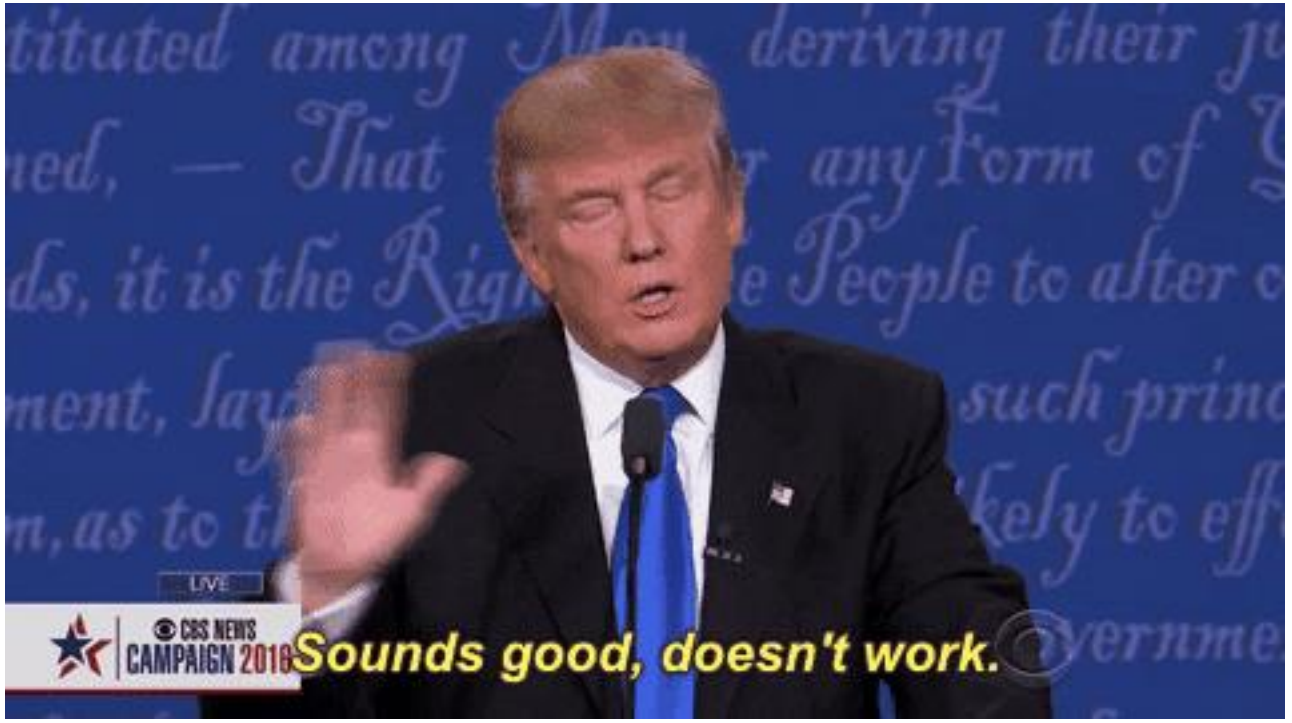
ROW		ENTRY	ENTRY
3	Column 0 begins	11	0
0		41	2
1	Column 1 begins	22	3
2	Column 2 begins	33	5
3		43	8
2	Column 3 begins	14	
3		35	
0		44	
1	Column 4 begins	25	

But why ?

- As the feature values are sorted, split candidates can be easily computed
- Blocks are compressed
- If data does not fit into memory it is divided into multiple blocks and only one block is loaded at a time
- Easy to enable

```
dtrain = DMatrix('my_big_file.train#dtrain.cache')
```

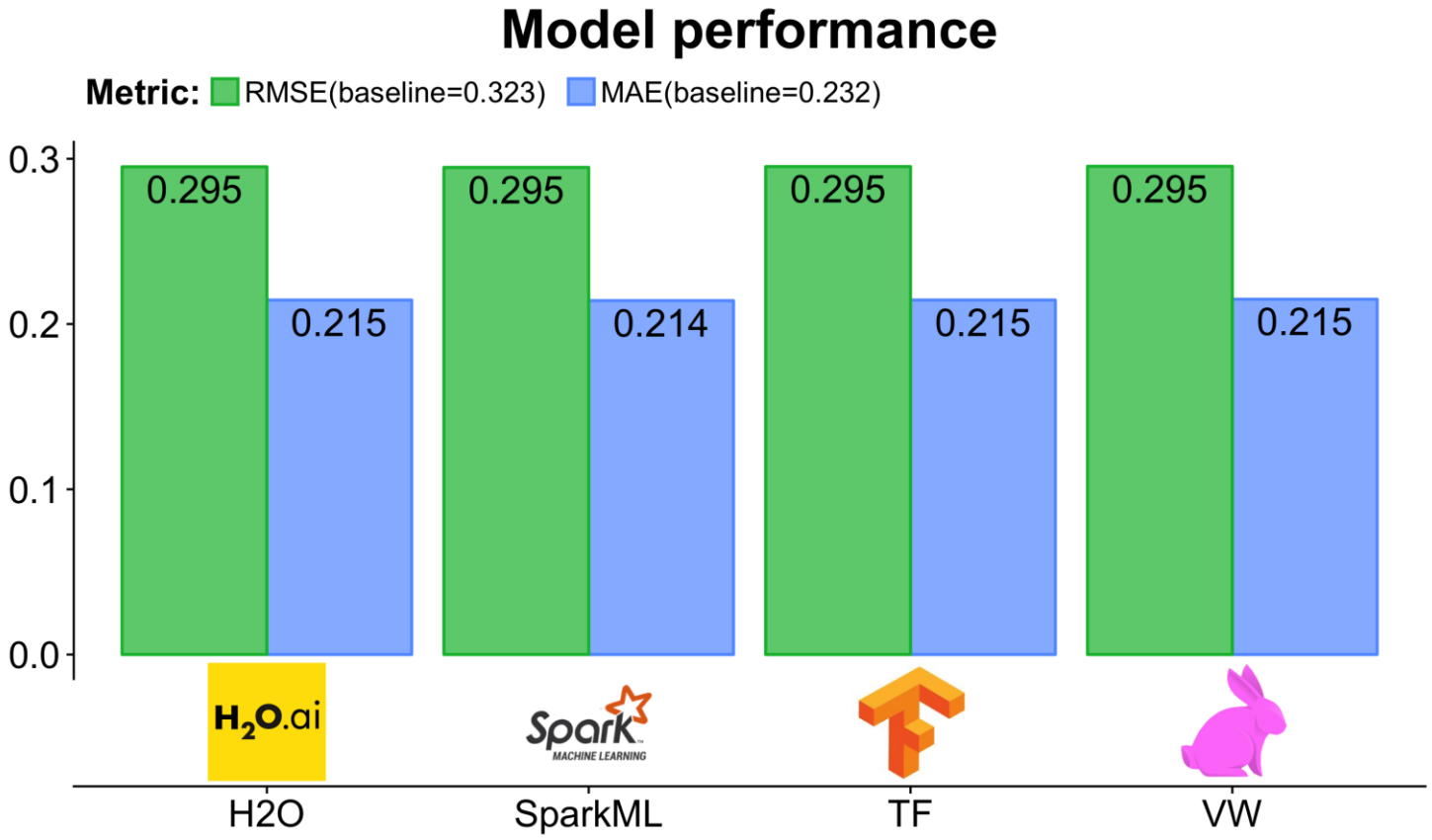
Benchmark time !

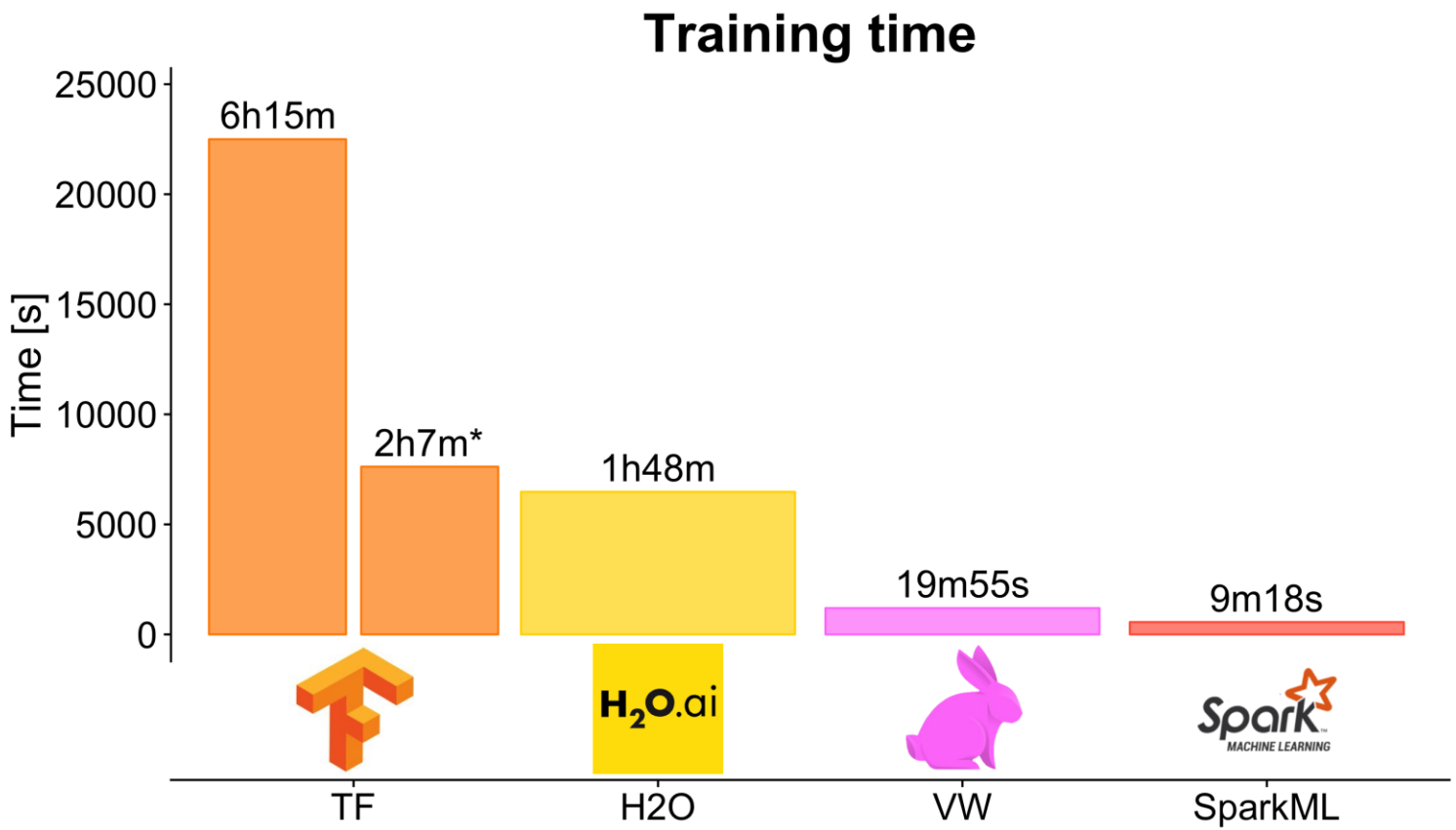


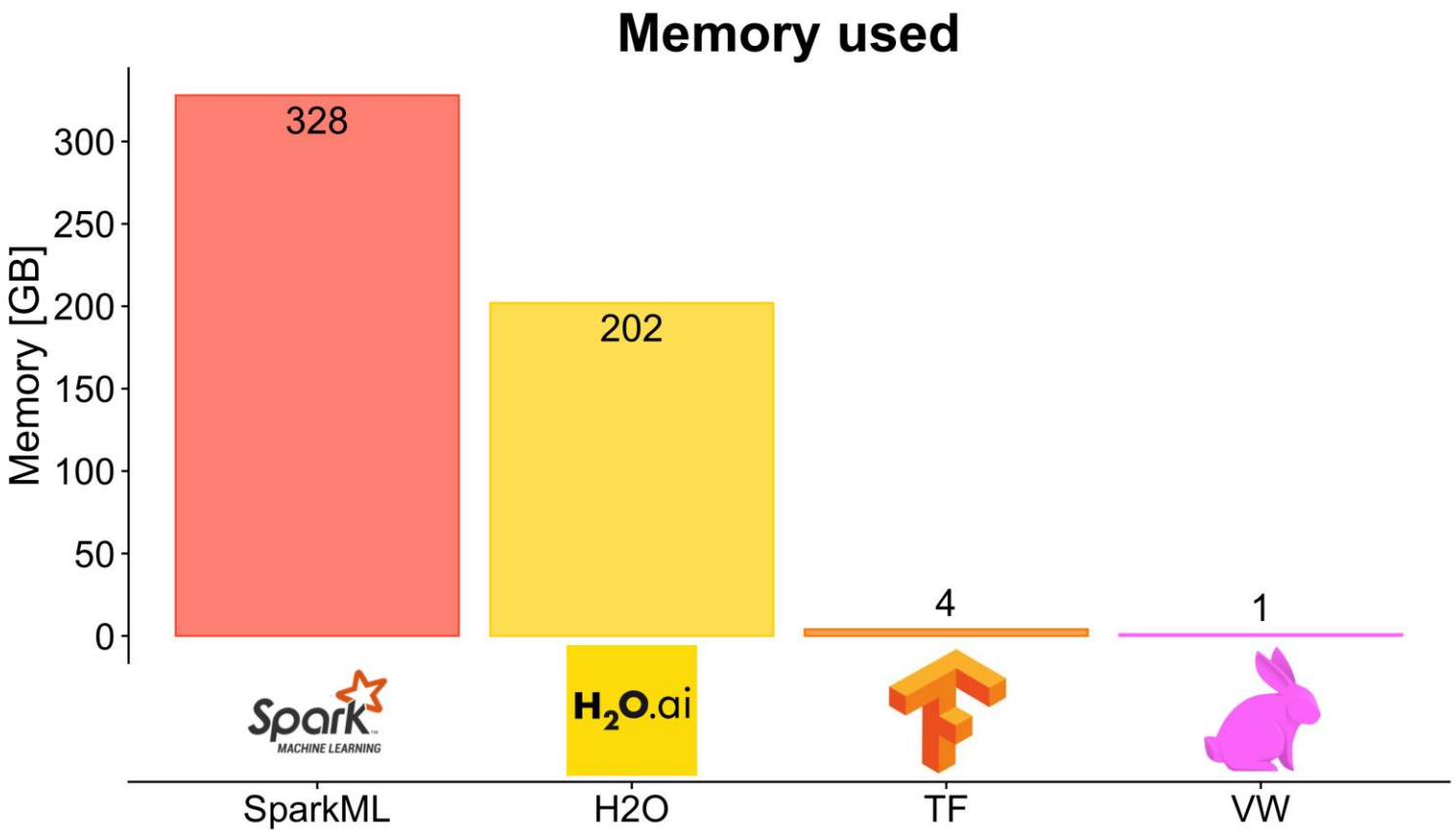
Experimental setting

- Linear regression 490M data points (~20GB)
- Comparison of H20, SparkML, Tensorflow and Vowpal Wabbit

Source https://booking.ai/crunching-big-data-with-4-machine-learning-libraries-284ae3167885


Model performance
Metric: RMSE(baseline=0.323) MAE(baseline=0.232)
0.3
0.2
0.1
0.0
0.295
0.215
0.295
0.214
0.295
0.215
0.295
0.215
H2O
SparkML
TF
VW


Training time
Time [s]
25000
20000
15000
10000
5000
0
6h15m
2h7m*
1h48m
19m55s
9m18s
TF
H2O
VW
SparkML


Memory used
Memory [GB]
300
250
200
150
100
50
0
328
202
4
1
SparkML
H2O
TF
VW

Bottom line

Library	Algorithm	Time	Memory	Cores	RMSE	R^2	MAE	Model size
H2O	L-BFGS	1h48m	202 GB	13	0.2952	0.3161	0.2145	13M
SparkML	L-BFGS	9m18s	328 GB	65	0.2948	0.3179	0.2142	2.1M
TensorFlow	FTRL	6h15m (2h7m)*	4 GB	5 (5 read, 1 train)	0.2952	0.3155	0.2145	3.3M
VowpalWabbit	SGD	19m55s	1 GB	2 (1 read, 1 train)	0.2954	0.3147	0.2150	2.2M

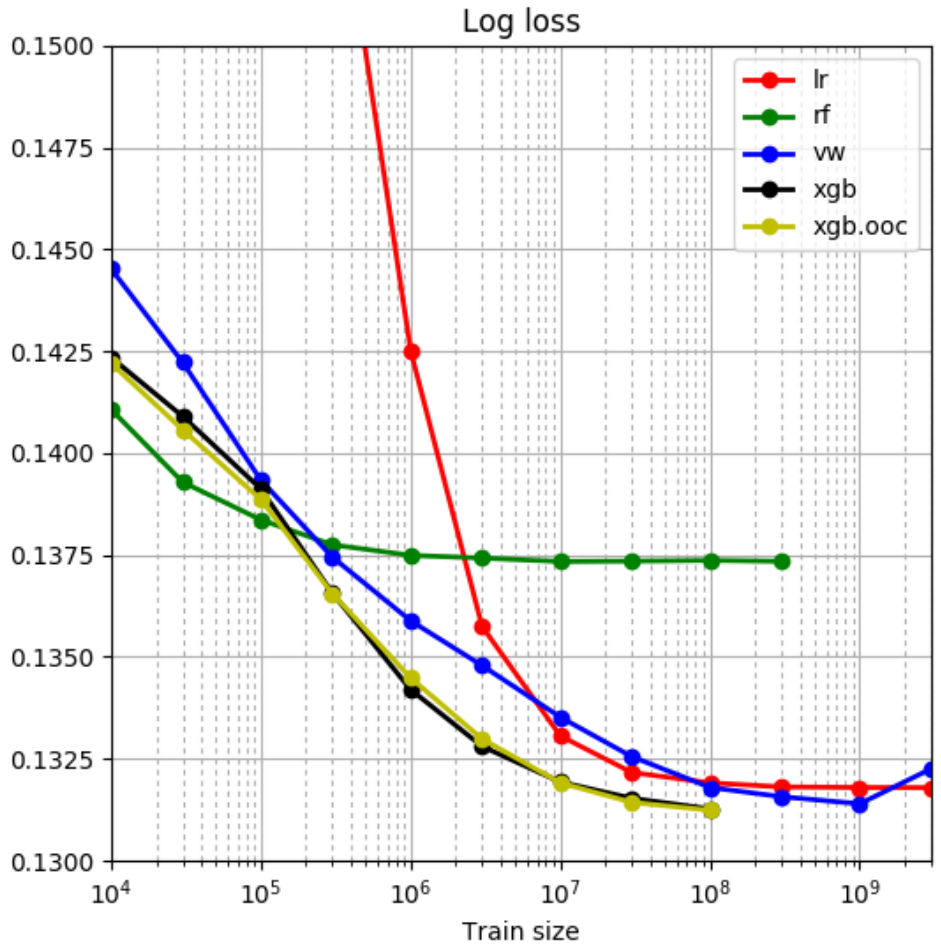
Another benchmark

- Criteo 1TB dataset (4B rows)
- Local models were trained on a 12-core (24-thread) machine with 128 GiB of memory.
- Distributed training:
  - 256 cores and 1 TiB of memory for training on datasets up to 300 million of lines
  - 512 cores and 2 TiB of memory for training on one billion and 3 billion lines of train data.
  - 4 cores and 16 GB of memory per Spark executor was used.

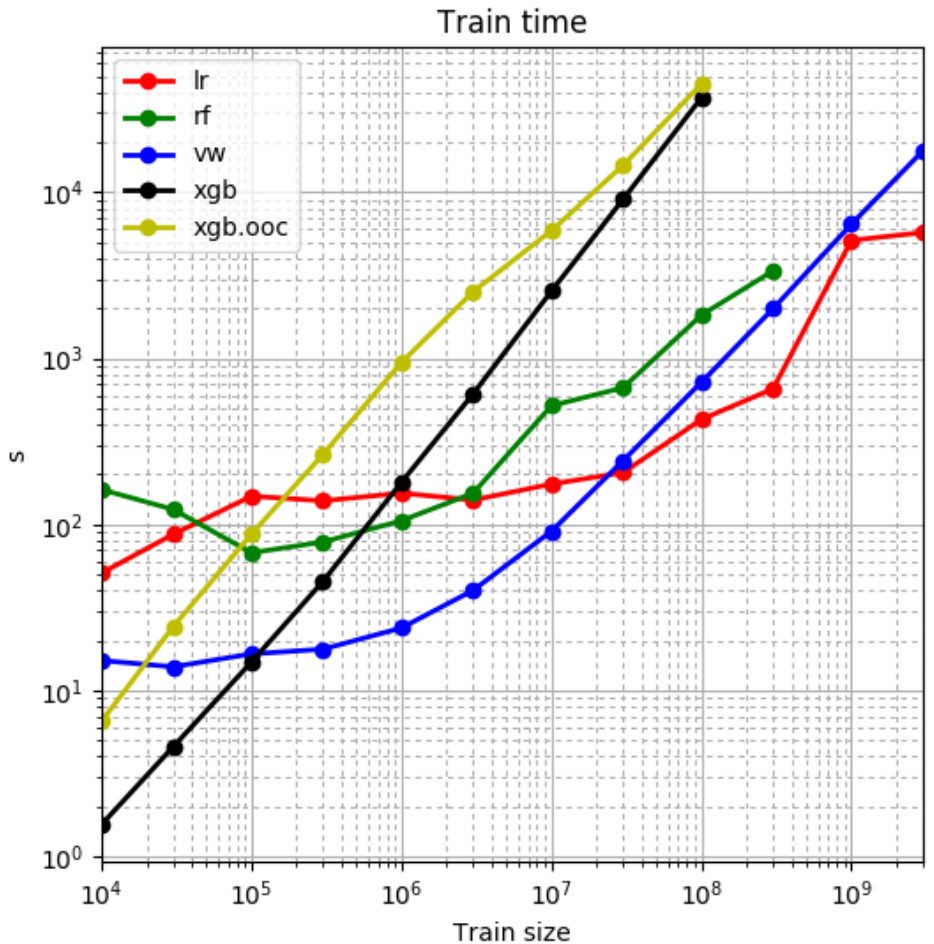
criteo

Source https://github.com/rambler-digital-solutions/criteo-1tb-benchmark

Model Performance



Model Performance



So Large Scale ML on one machine is possible ?

So Large Scale ML on one machine is possible

- Yes ! And there are quite a few libraries that can help you

- But your preprocessing is taking too long now !
  - Task specific code that is less optimized compared to a general purpose ML library
  - This can become a bottleneck quickly

Very simple parallel processing in Python

- Imagine you want to apply some preprocessing to each line of your data

```
def preprocess_line(line: str) -> str:
    return #Some complex preprocessing

with open(filename) as f:
    preprocessed_lines = map(preprocess_line, f.readlines())
```

Very simple parallel processing in Python

- **An immediate way to make it ~8x faster**

```
import multiprocessing

with open(filename) as f:
    with multiprocessing.Pool(processes=8) as p:
        preprocessed_lines = p.map(preprocess_line, f.readlines())
```

Very simple parallel processing in Python

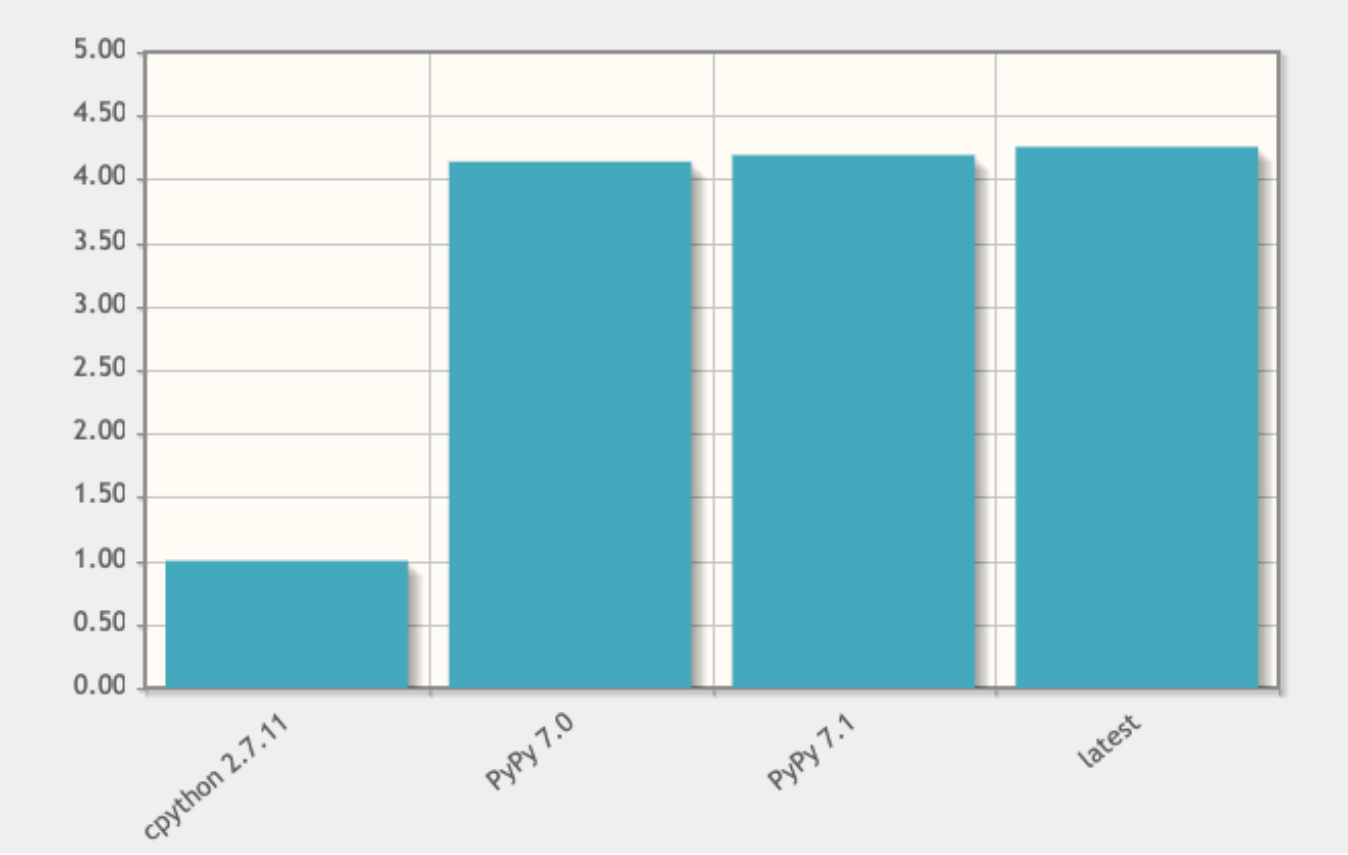
- Parallel processing is important
  - Use all the cores of your computer
  - Explore the multiprocessing lib in python

- Most preprocessing steps are embarrassingly parallel
  - Applied line by line
  - They can even be distributed
    - A Criteo example: Preprocessing in Spark, training in Tensorflow

Can we just make Python faster ?

- Pypy https://www.pypy.org/
- A fast, compliant alternative implementation of Python
- Uses just in time compilation
- Use it only for large, slow-ish programs
- What's the catch ? not all libraries are compatible

A Faster Python



Other ways to make Python fast

- Cython (https://cython.org/)
  - If C and Python had a baby
- Numba (http://numba.pydata.org/)
  - Decorate your way out of slow code
- Writing C modules
  - If you love excruciating pain

Other ways to make Python fast

- Cython (https://cython.org/)
  - If C and Python had a baby
- Numba (http://numba.pydata.org/)
  - Decorate your way out of slow code
- Writing C modules
  - If you love excruciating pain

I hope this helps

I hope this helps

- Sometimes the best way to do Machine Learning on Big Data is actually not to do it
  - Your data may be smaller than you think after preprocessing and sampling
- Always try to start with a single machine solution and set up
- There are a lot of iterative, out-of-core algorithms in your favorite ML library
  - Loading data into RAM is not the only path
- Preprocessing will quickly become the bottleneck, speed it up !

Wrap-up (1/3)

Large-scale machine learning is not about peta-bytes of data

It is about making complex tradeoffs due to time constraints (model complexity, number of examples, optimization algorithm, etc.)

Start with a known optimizer (e.g. Adam)

The library of existing models is huge (don't be a hero)

Focus on the data, the task and the metrics

Wrap-up (2/3)

There are amazing frameworks and hardware out there.

There are built by real humans. They have real bugs.

Learn to read the docs and stackoverflow.

Wrap-up (3/3)

Your data may be smaller than you think (after pre-processing)

Always try to start with a single machine solution and set up

There are a lot of iterative, out-of-core algorithms in your favorite ML library (loading data into RAM is not the only option)

Preprocessing will quickly become the bottleneck, speed it up !