Heuristics for a Successful ML Solution

Eitan Farchi

Raviv Gal

Orna Raz

Marcel Zalmanovici

IBM Research, Haifa

What will you learn in this workshop?

Heuristics to introduce non-parametric statistical control into the ML solution development phases

- Never use single numbers for random variables
- Keep test-data separate from training data
- Automate and check for changes in training vs. deployment data distribution characteristics
  - Ex. Single features, single records, relations among features

Random variables examples

- The performance of your model
- The error in your training data

Emphasis on data driven requirements and design

Emphasis on testing at all phases, including deployment

Workshop highlight – mapping heuristics and statistical control to the development phases

Requirements and Design Heuristics

- Identify a high value business challenge
- Define business value metrics
- Ensure there is sufficient relevant labeled data

Development Heuristics

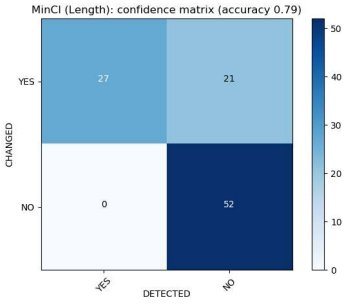
- Introduce automation and statistical control into the data pipeline and hyper parameter tuning

Test and Deployment Heuristics

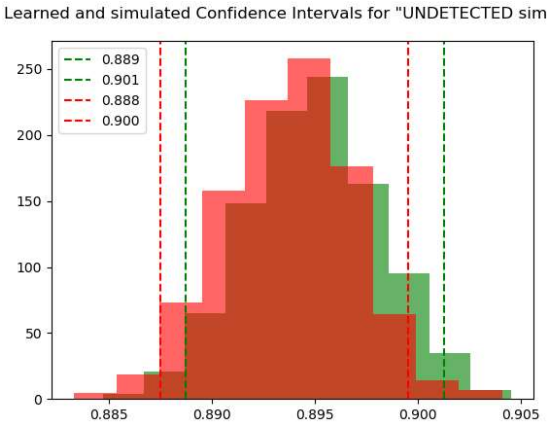
- Make sure there is a test set never used for training
- Identify random variables such as model performance and apply statistical control

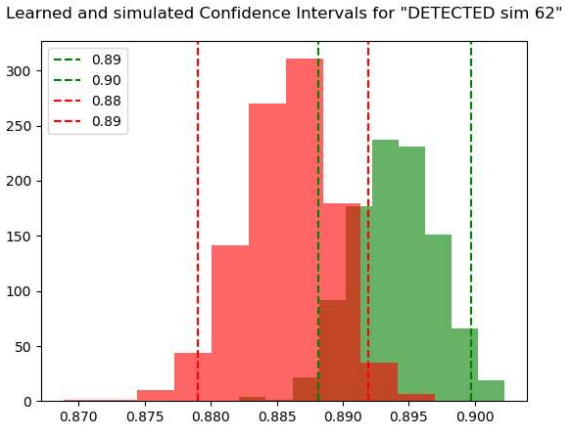
Move away from averages to confidence intervals

Workshop hands-on



- Requirements: Python 3
- Extract course.zip and run the *.py files per the pointers under the various heuristics
- What our framework does: identify statistical relations in the data and determine if they break
- How
    1. User needs only to provide ‘analyze’ and ‘noise’ methods per a random variable
    2. Automatically compute non-parametric confidence interval for the random variable
    3. Simulate deployment with and without drift and compute a confidence interval per iteration
    4. Determine similarity of confidence intervals (non-parametric test)
    5. Compute accuracy of analyzer (confusion matrix)





Pre-requisites

Working knowledge in

- Machine learning (ML)
- Statistics
- Software Engineering (SE)

Definitions of data sets in use throughout learning

Training Set

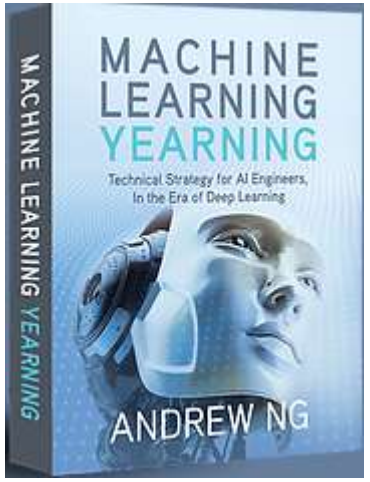
- On which you run your learning algorithm

Dev (development) Set

- Used to tune parameters, select features, and make other decisions regarding the learning algorithm. Sometimes also called the **hold-out cross validation set**

Test Set

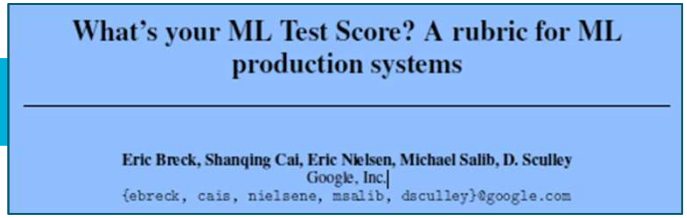
- Used to evaluate the performance of the algorithm, but not to make any decisions regarding what learning algorithm or parameters to use



Choose dev and test sets to reflect data you expect to get in the future and want to do well on

Andrew Ng, Machine Learning Yearning. Draft (2018). Chapter 5

Best practices examples – not part of this workshop



Training

- Future development – New features can be added quickly
- Training is reproducible – Can be used for testing
  - Comparing results with new feature / bug fixing
- Feature engineering should have Unit Testing

Udacity

Coursera

available online

DataCamp

Model

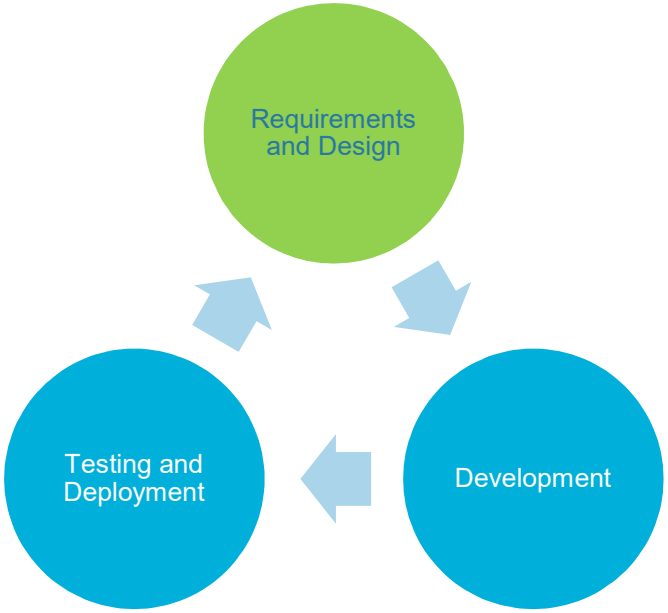
- Model development – Selecting among potential models
- Model training (e.g., k-fold cross validation)
- Models can be quickly and safely rolled back to a previous serving version
- Model performance monitoring – training speed, serving latency, throughput, or RAM usage

Deployment

- Deployment and infrastructure sanity checks – Canary testing, A/B testing

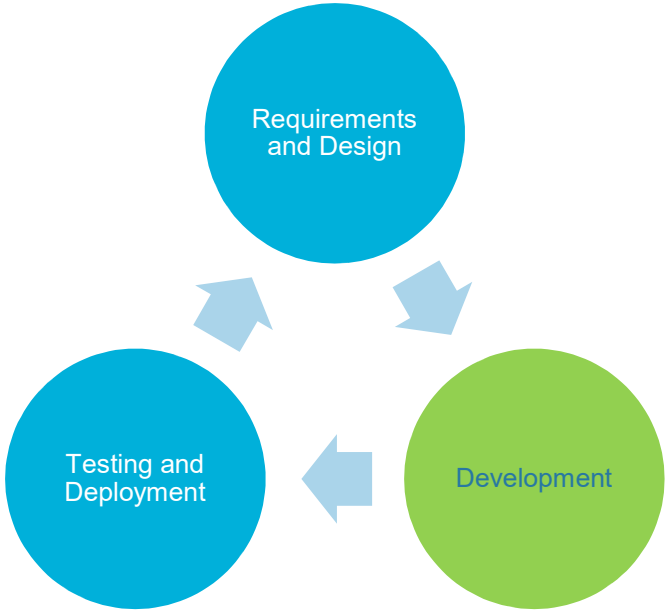
ML checklist
Requirements and design

- System business value metric is defined and a test set not used in learning is obtained
- Training data represents real life data
- There is 10 times more data than features if a generalization model is to be obtained
- Per-labels there are at least 30 examples
- Labeling accuracy is statistically estimated
- On going labeling is possible



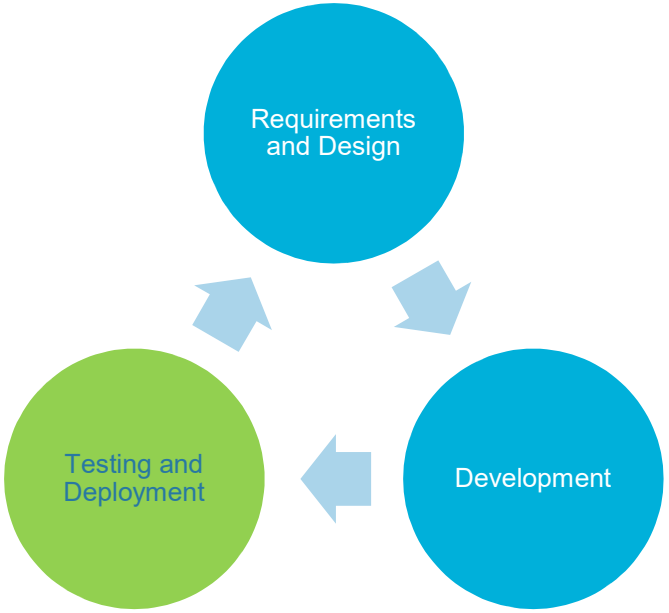
ML checklist Development

- Relearning from raw data is automated (the entire data pipe line is automated)
- For an object, a mapping that obtains its characteristic for each stage of the pipeline exists
- A complete list of hyper parameters is available. For each hyper parameter determine if its optimal value was searched for automatically as part of the automatic data pipeline. Each hyper parameter that was not searched for increases the risk
- Model was developed using advanced statistical techniques that estimate its performance (at least k-fold cross validation)
- Non parametric confidence intervals should be developed for all averages
- Statistical assumption made during the data pipeline development are made explicit and an appropriate check is implemented at deployment time. E.g., two features are correlated and one of them is dropped. At deployment time we check if the two features are still correlated



ML checklist
Testing and deployment

- Testing was done on the ground truth that was never used for learning using advanced statistical techniques.
- At least boot strapping and non-parametric confidence intervals
- Ground truth is sliced to identify weakness in the system (Asset under development)
- Statistical assumptions developed at the development stage are being checked and appropriate alerts are in place if they fail
- On going labeling is implemented



Requirements and design heuristics

Explicitly measure business value. Don’t Confuse ML metrics with business value

Is the data relevant for the business objective?

Identify and select all relevant data (structured and otherwise)

Data must be of valuable volume - ML is not the only solution

Is data adequate for learning?

Is the training data representative of deployment data

Get to a supervised learning problem

Requirements and Design Heuristics

- Identify high value business challenge
- Define business value metrics
- Ensure there is sufficient relevant labeled data

Development heuristics

Automate the data pipeline from raw data

Capture statistical assumptions resulting in feature selection/augmentation/reduction and cleansing

Design and automate experiments to find the best values for hyper-parameters

Create a baseline to compare your model with

Development Heuristics

- Introduce automation and statistical control into the data pipeline and hyper parameter tuning

Test and deployment heuristics

Provide confidence intervals rather than a single number - use non parametric statistics and bootstrapping to test the system

Test the solution on the raw data

Apply statistics throughout the solution lifecycle. Make sure statistics are used not only when building a model, but also when deploying it

Implement a feedback mechanism for users to help fine tune the solution – continuous labeling of data

Estimate the probability of a labeling mistake and take it into account in testing (and development)

Introduce deployment monitors

Test and Deployment Heuristics

- Make sure there is a test set never used for training
- Identify random variables such as model performance and apply statistical control

The nature of the heuristics –
a lot of the value can be gained by following simple heuristics

The nature of the heuristics varies

To apply them successfully

- Some require mathematical and/or statistical maturity (M)
- Others require programming capability (P)
- Others require data analysis capabilities (D)
- Yet others are soft experience based

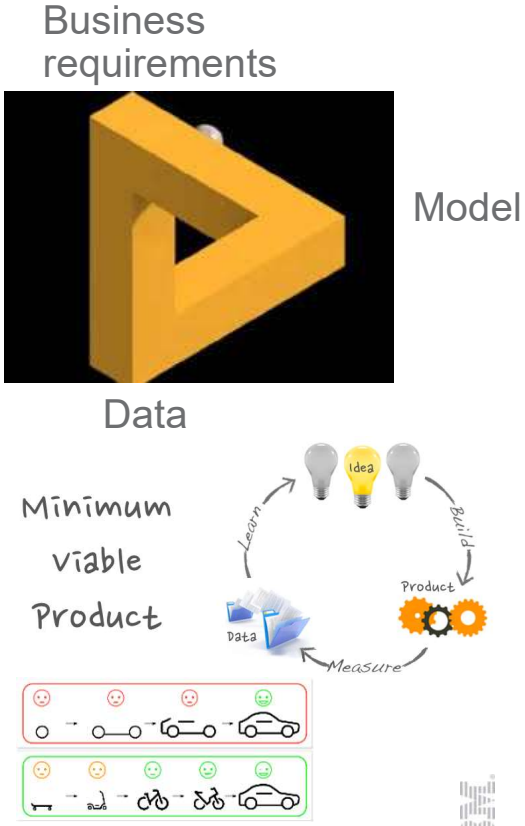
There is no correlation between the skills required for a given heuristic and its value…

I'll highlight the required skills to facilitate their consumption

REQUIREMENTS AND DESIGN

Is the data relevant for the business objective?

- Design time - Marry business requirements with relevant data that can be successfully learned through appropriate optimization objectives
- Define potential business objective with SME (Subject Matter Expert), data scientist and architect
  - MVP (Minimum Viable Product) approach: small iterations to both define business value and understand what value we can get
- Decision of objective depends also on the data!
  - Data may not include significant signal supporting the objective
  - Ex. Classification of software defects. ODC (Orthogonal Defect Classification) vs. custom classification
  - Data may have a signal that is ‘too strong’
  - Ex. It is always the case that a defect opened by a manager is of highest importance. No sense in learning this. Use a rule instead



Is the data relevant for the business objective?

Define desired business value of the ML solution including sanity check of feasibility

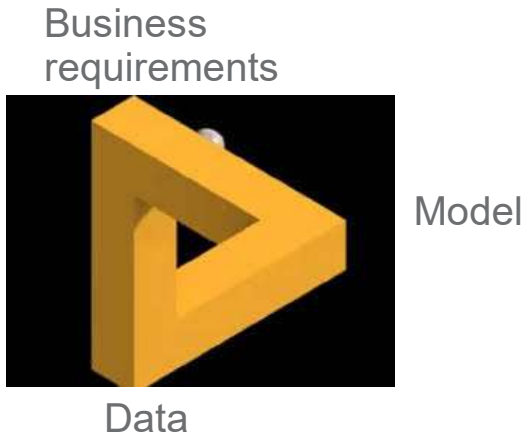
- Interview SME to crystalize main challenges. For each identified challenge:
  - What is the bottom-line added value?
    - User story – design thinking
  - DATA / LABELED DATA
    - What data is needed – does it exist and how hard would it be to exploit
    - Rough estimation of consolidation and cleansing effort
    - You must actually review data at this stage! Don't imagine, verify
    - Ex. imagining defect mapping to tests when test are run in batches
    - Define acceptance criteria
    - Must allow for false positives (FP) vs. false negatives (FN)
  - How will you validate/estimate success? (see explicitly measure business value)
    - Define value metrics
    - To be complemented with solution implementation metrics (e.g., F1-score)
  - Define other extra-functional criteria: Performance, Responsiveness, Resource limitations, Data requirements, such as online or real-time utilization

How to get ideas from SME:

- Data mining to automatically suggest interesting signals and characteristics of the data
- Get SME feedback
- Get SME additional suggestions for questions
- Check if there is relevant data for learning

Explicitly measure business value
Don’t confuse ML metrics with business value

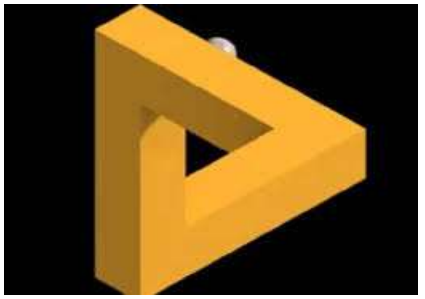
- Measure business value in addition to ML metrics
  - Ex. A recommendation system. Optimization objective: classify user query (voice or text) to list of products
  - ML metric: accuracy
  - Business objective: increase user purchase of products
  - Is user click rate a good measure? Actual purchase a good measure?
  - Ex. A hazard notification system. Optimization objective: classify text document into hazard yes/no
  - Business objective: improve notification speed while requiring less human effort and with comparable accuracy
- When validating and throughout deployment
  - Map metric back to business requirements
  - Test metric on: entire data & data bins that represent requirements
    - Ex. Various types on incoming text documents



Explicitly measure business value – integrating with a legacy rule base system

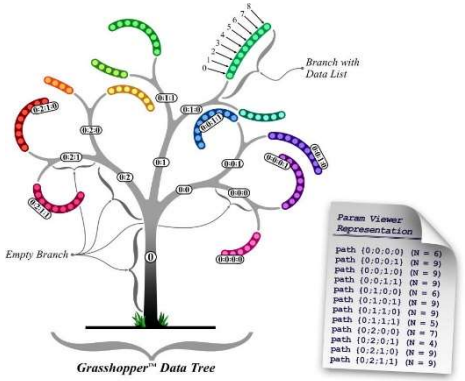
- Ex. A rule base system is operational and translates DB entities to governance categories such as customer personal record
- The ML is required to improve the mapping
- There are various ways to integrate the ML and the rule base system:
  - A condition should determine if the rule base is applied or the ML
  - The ML can be applied and if it predicts certain categories then the rule base is applied. Ex. a category called ‘other’
  - Weighted decision of the ML and rule base prediction
- Instead of developing the ML standalone, follow the same best practice applying the development set and test set but
  - Measure the different potential integration strategy instead of just the ML performance
  - Use the hybrid system performance on development and test set as a baseline
- Your recommendation should include a quantitative recommendation on how to integrate the rule base and ML

Business requirements



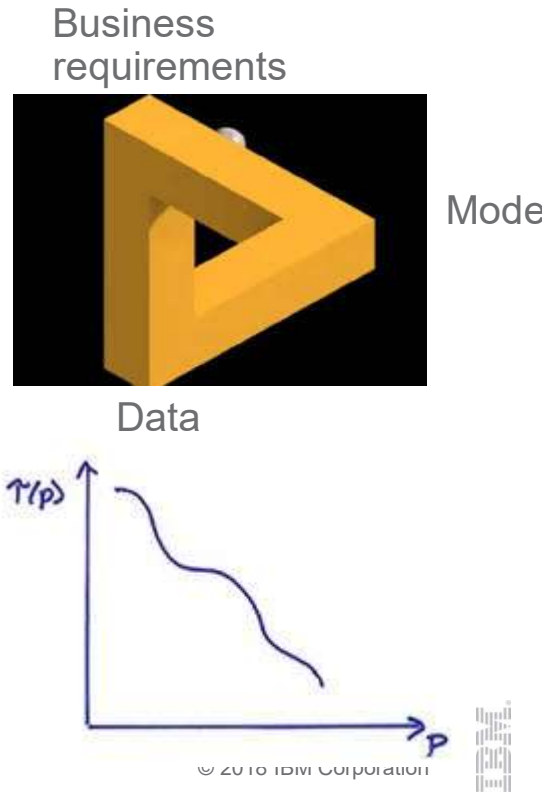
Model

Data



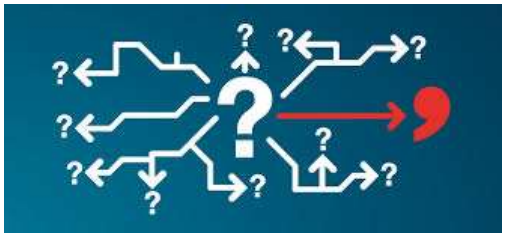
Explicitly measure business value – system is expected to start weak but continually improve

- What if the ML is built from no real data?
  - Ex. at operational time more and more examples are expected
- Different metrics of success are needed!
  - Over-time simulation rather than point-in-time estimation
- Instead of developing a model and measuring its performance (Ex. accuracy) simulate the arrival of data
  - Given the training set sample without replacement data and feed it to the learner
  - The learner predicts and then gets the true label
  - At the end of the simulation measure the percentage of mistake at consecutive intervals of the simulation – they should be decreasing
  - You can apply regression to prove that this is the case
  - Repeat the time simulation several time
- The business value is that the system is improving at a desired rate when new data is arriving!

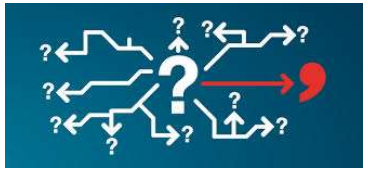


Identify and select all relevant data

- At design time of an ML solution you will typically consider some desired business value and a given source of data
  - Ex. Given 1000 tickets (problem reports) try to determine the nature of the problem automatically
- Don't accept the given data as the only source of information. Instead, explore other possible sources through the implicit, meta data, and traceability heuristics:



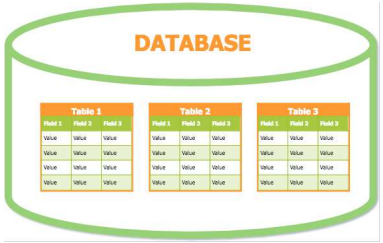
Identify and select all relevant data – examples of the implicit meta data and tractability heuristics

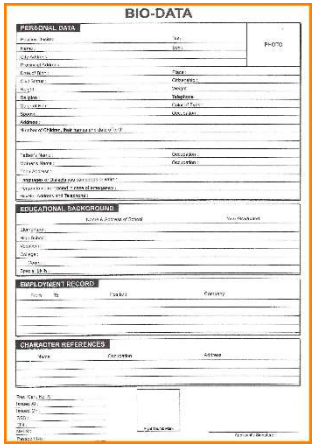


- The tickets may be organized in a filesystem and filenames may be relevant (implicit)
- The ticket is "meta data" and it may point to logs/attachments that are relevant
- Traceability – other systems may contain additional relevant information that needs to be obtained through traceability
  - Ex. If the ticket represents a software problem then information on the parts of the code that was changed is relevant
  - Understanding the organization process that applies to the data is important
- Records of a relation DB table may have missing fields that can be obtained from other tables through a join operation (implicit)
- Stock market outcomes impact traffic accidents near Wall St.? (implicit)

Use all relevant data: Do NOT ignore structured data!

- Structured data is good!
  - The structure typically conveys information that can be used for learning or used in addition to learning BUT
  - May pose challenge to the data representation
- Do not ignore it just because you want to do learning
  - Ex. Form text fields are converted to bags of words
  - Ex. Database tables are flattened and relations learned
- Find a representation that captures the structure
- If needed, use rule-based logic to identify the relevant structure of your input
- Add it to the learning solution! (hybrid approach heuristic)
- This way you do not lose potentially high-quality information





Data must be of valuable volume - When not to use learning

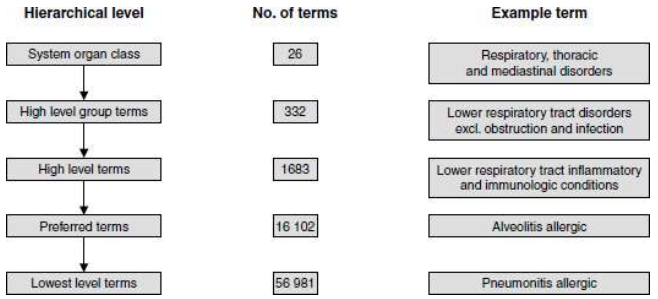
- Learning requires sufficient data
- Rules of thumb for data sufficiency
  - Size of training data at least 10x the number of features
    - Based on PAC (Probably Approximately Correct) learning
  - Number of examples per label
    - At least 30; this is very small – assumes that we know the data distribution (which we don't) and that it is normal
    - Think about a binary classification for that label
  - Gaining confidence about a very small improvement in performance requires more data as (you need to gain statistical confidence that the small improvement is not random).
- See - is the training data representative of the deployment data?
- Do not assume you will have sufficient data – measure and verify it! (see how to estimate data volume heuristic)
- Remember that you also need data for testing
  - When training – development set – used multiple times
  - When testing – test set – you cannot use it to change your model! If you do, you will need more data for testing
- Remember labels
    - You cannot learn labels that are not in the training set
    - Need to understand and control labeling mistake rate (see Estimate the probability of a labeling mistake)

MedDRA Hierarchy example

Hierarchical level	No. of terms	Example term
System organ class	26	Respiratory, thoracic and mediastinal disorders
High level group terms	332	Lower respiratory tract disorders excl. obstruction and infection
High level terms	1683	Lower respiratory tract inflammatory and immunologic conditions
Preferred terms	16 102	Alveolitis allergic
Lowest level terms	56 981	Pneumonitis allergic

Data must be of valuable volume - How to estimate data volume

MedDRA Hierarchy example



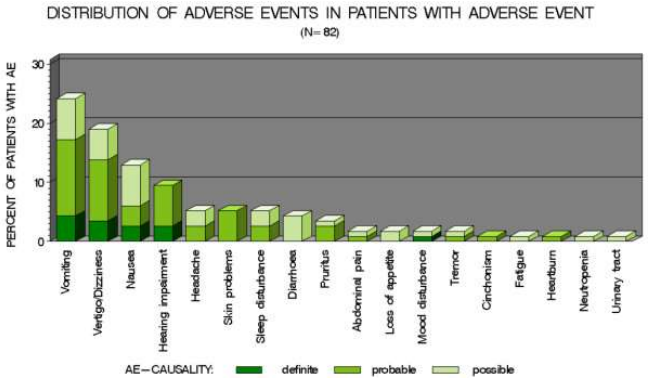
- Even if the data is collected from multiple locations and in multiple formats, go through this effort and measure how much data you have per each label
- How many unique labels are there?
- Are there labels with only a few data points?
- Draw a histogram and the average. Are most labels above or below?
- Often, there are heavy tailed distributions and not enough data in the tail (see handling heavy tailed distributions and labels with few data)
- Label accuracy (see estimating the probability of labeling mistake and taking it into account)
  - For images, text, audio – are you (or an SME) able to correctly label them?
  - If manual matching is challenging, expect AI to have trouble as well
- How to count? We may need diverse examples for the same label
  - Ex. Context around a text phrase, phrase near matches
- Too many identical/similar data points -- should count as one/few
- Too diverse data points AI cannot generalize

Data must be of valuable volume - How to handle few data and heavy tailed distributions

- Distinguish between having sufficient data per label and having few data per label
  - Ex. Heavy tailed distribution, meaning that most of the probability mass is on the left of the diagram
  - Ex. MedDRA has tens of thousands of labels and therefore we expect a heavy tail phenomenon on the training set
  - The hierarchy represents medical adverse events so you expect very few examples for the more severe cases
- Binary classifier per label to estimate data sufficiency for this label (precision and recall)
  - If that does not work, insufficient data for learning this label
- Few data: use a hybrid approach
  - Rule-based logic for the part that has few data
  - Learning for the part that has sufficient data

MedDRA Hierarchy example

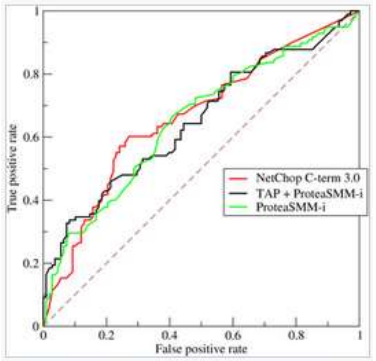
Hierarchical level	No. of terms	Example term
System organ class	26	Respiratory, thoracic and mediastinal disorders
High level group terms	332	Lower respiratory tract disorders excl. obstruction and infection
High level terms	1683	Lower respiratory tract inflammatory and immunologic conditions
Preferred terms	16 102	Alveolitis allergic
Lowest level terms	56 981	Pneumonitis allergic



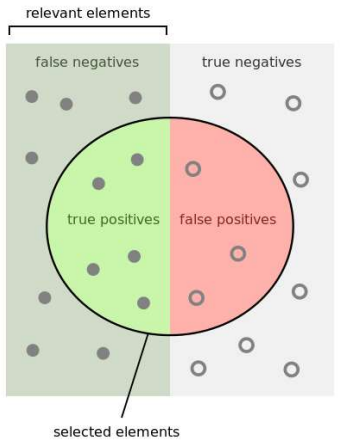
Data must be of valuable volume - Data, labels, and ML performance metric interplay

Even when you have sufficient data per label:

- Metric should match the data characteristics
- Beware of getting results that are 'too good'
  - Ex. Great accuracy because always classifying as most common class
    - Ex. 99.5% accuracy for breast cancer detection with 0 recall (always predict 'no')
  - Consider other metrics: precision/recall/F1-score/ROC (receiver operating characteristic) curve
  - Consider weighing the error types per their cost
- Balance labels to have a more uniform distribution
  - Over sampling/ under sampling
- Increase weight of the labels with fewer examples
  - AdaBoost goes over an ensemble of weak learners to change the input distribution in order to give more weight to labels that get poorer results



ROC curve of three predictors of peptide cleaving in the proteasome.



How many selected items are relevant?

Precision =

How many relevant items are selected?

Recall =

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Is data adequate for learning? A hybrid approach

- Identify areas of data with sufficient data for learning and areas with insufficient data for learning (see how to estimate data volume)
  - Ex. Some labels may only have a couple of examples
- Make this a design point: be able to quickly communicate to your client where there is insufficient data for learning
- What can you do with insufficient data for learning?
  - Try to create sufficient data: Over sampling? Synthetic data? Aggregating multiple labels into a single meta-label to get sufficient data?
    - Ex. If you have a labeling hierarchy, can you go up the hierarchy?
    - Ex. When you have any other structure, such as DB table indices or file system directories, can you aggregate according to this structure?
    - Consider aggregating all insufficient data into a single label 'other'. Then use a non-learning approach for that data
  - Try to provide a rough estimate solution
    - Ex. If you need to classify free-text terms into products, and you only have 1 example per product, consider a dictionary approach plus some Natural Language Processing (NLP) for new texts. If none of the dictionary terms fit, notify the user, consider asking the user to label (select a category) the data

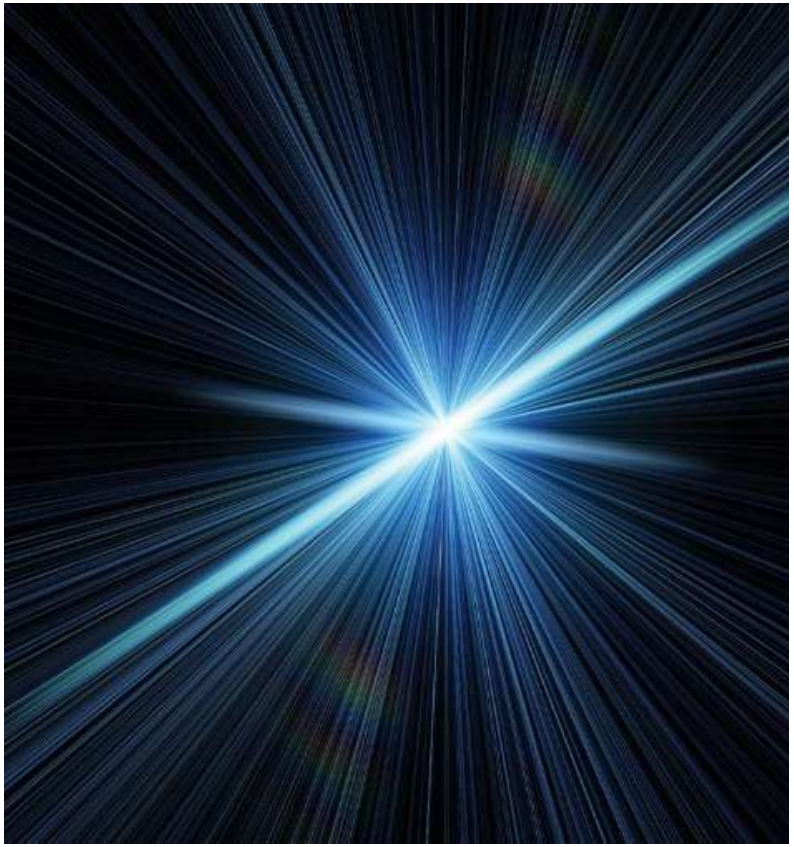
Is the training data representative of deployment data?

- Assumption underlying statistical machine learning: the training and test data distributions represent the deployment distribution
- The data is collected in a totally different way than the data the system will encounter during deployment
  - The self-driving car is trained in different weather conditions or in different countries
- Data collected is different due to problems in measurements or different scales
  - Missing data
  - The scale is different (camera is positioned higher)

- Use knowledge about the data distribution to determine if the data meets reality
  - Domain knowledge
    - If you know that percentage of fatal side effects is negligible and you get data that has 80% fatal side effects, then something is wrong with the data
    - Breast cancer percentage in the population is around ½%
  - Statistical relations on data
    - Capture and monitor feature relations and behavior (min/max, correlation,…)
- Hands-on: Run min.py a few time to see the identification of changes around min

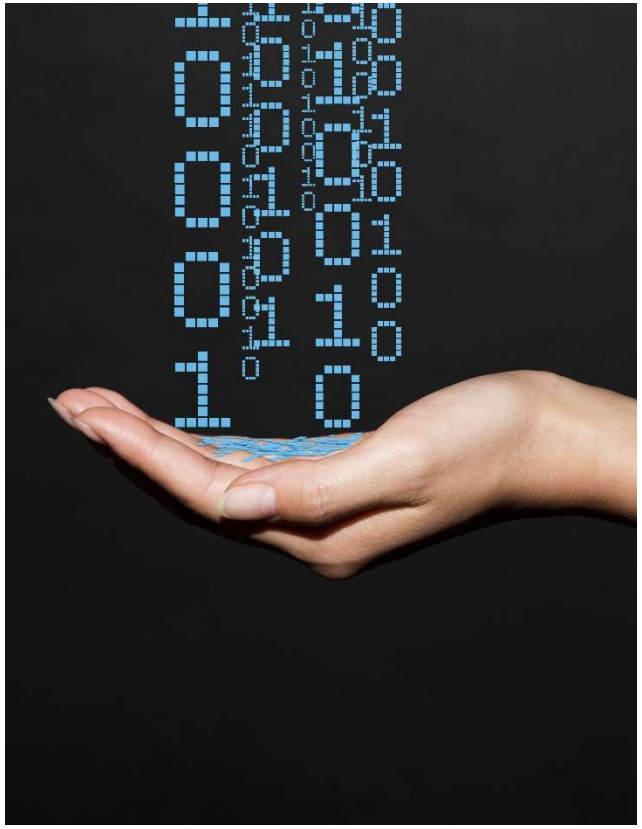
Get to a supervised learning problem

- ML shines when supervised learning can be applied
  - State of the art, it's more mature and has a stronger success record
  - It's an indirect indication that you are able to quantify a business objective (more on that later) that will be used in testing the system
  - Accurate labels are required for training and testing purposes
- The challenge is to identify appropriate labels for the data
- Two general heuristics apply:
  - Choose a subset of the attributes in the data for the labels
  - Create an experiment that will generate desired labeled data
- To predict system performance, a labeled test set will be required



Create an experiment that will generate desired labels

- Goal: Develop technology that will identify network failures from the server logs
- Define some general client workloads and inject network failures artificially at runtime
- Save the server logs for each run of our workloads
  - We now have server logs that are labeled with the type of network failures that were artificially injected



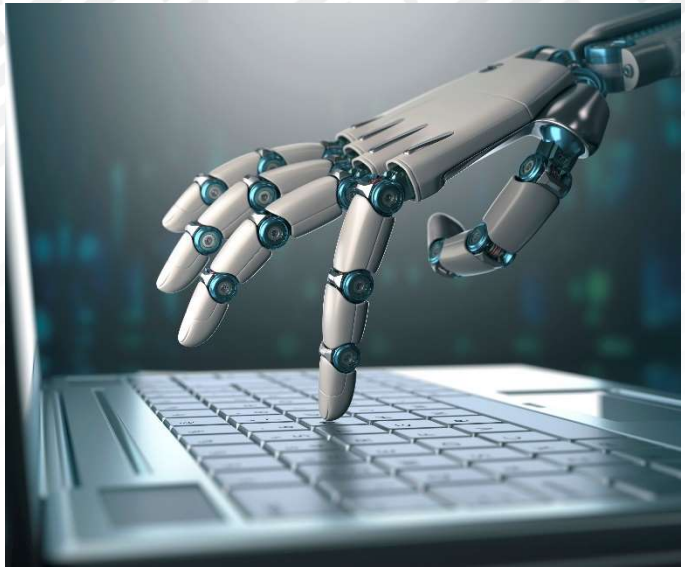
The devil is in the experiment - defining envelop of operation through appropriate experiment design I

- The experiment defines the requirement reflects the desired business goal, and envelop of operation of the system
  - Definition of the statistics over the test set reflects what we are attempting to measure
- Consider if the following is correctly expressed in the experiment
  - The system start with little or no data and need to improve over time
    - for a new group/person in the organization
    - for a new organization altogether
  - Example - need to complete the query a user is entering
  - Example - map business terminology of a specific organization to generic business terminology

The devil is in the experiment - defining envelop of operation through appropriate experiment design II

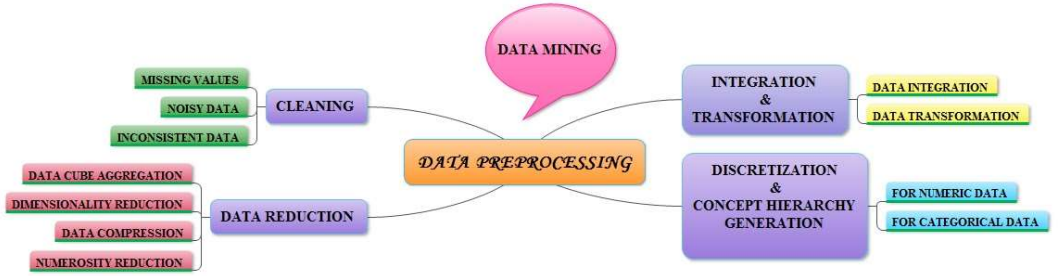
- The system combines machine learning and some rule to obtain the best result
  - Some of the classes does not have enough data for learning
  - There are a few alternatives being considered including legacy system that use no ML
- The objective function for the learning is just a mean to an end and does not reflects the business value
  - Need to define another experiment that does
  - Example - the model predicts if the next test will fail but is used to reduce the volume of running tests in the lab while keeping the same level of problem identification
- Critical for performance reasons - the same test data is used but different experiments are defined using it
  - A set of query templates can be used to calculate accuracy but also to simulate system reaction to new types of query

DEVELOPMENT



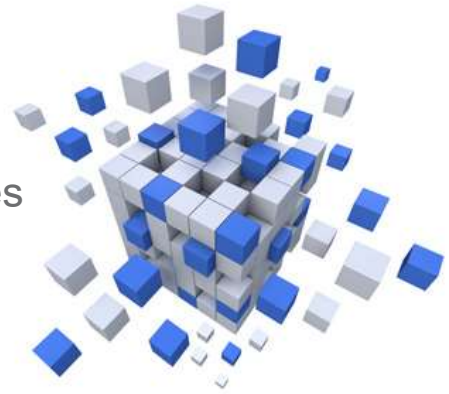
Automate the data pipeline from raw data

- Data preparation activities: Raw data needs to become analysis ready
- Feature engineering activities: Selection, augmentation, reduction
- Data cleansing activities: Realistically assume that data is ALWAYS imperfect
- Automate data preparation, feature engineering, and cleansing
  - See next slides for common activities
- Compare training data with deployment data see capture statistical assumptions resulting in feature selection/augmentation/reduction and cleansing



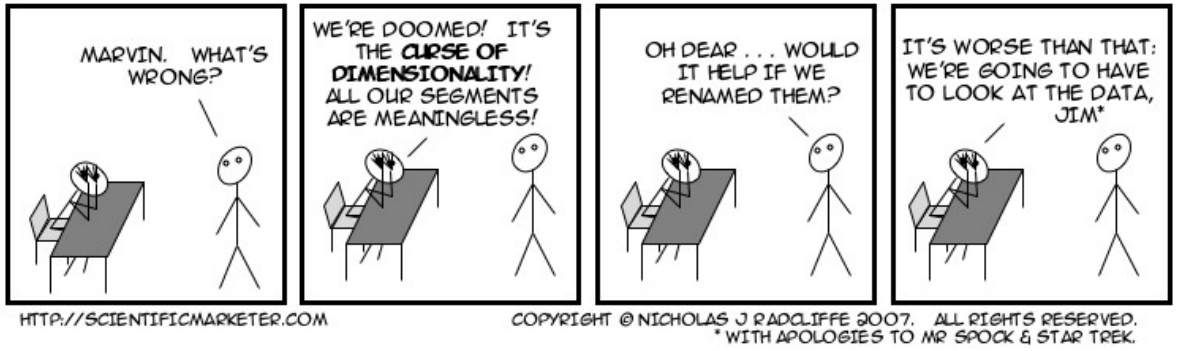
Data preparation and effort estimation (for feasibility PoC)

- Define required data retrieval
  - Offline, online, real-time
  - What is possible?
  - Time window requirements?
  - How much data would suffice for initial investigation?
    - For example: anomaly detection: months, including anomalies that you know how to identify
- PoC **Minimum** initial data intake
  - Partial data, no scalability, offline
  - For example: online and globally spread → CSV
- **Benefit**: estimate for the data preparation effort and
  - Sample data from each data source
  - Example merge of data
  - Minimal data for experimentation



Feature engineering

- Feature engineering is based on statistical properties of your training data
- Revisit feature selection, augmentation, reduction when data changes
  - Training vs. deployment
- Do these hold over your deployment data?
- Best to automate the process
- Ex. PCA (Principle Component Analysis)
  - Re-choose principle components and compare to training choice
  - How does this affect your model?

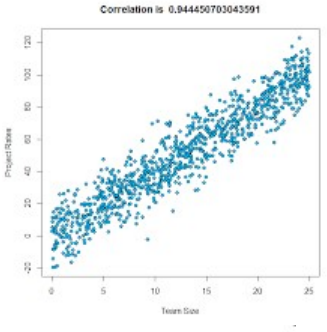


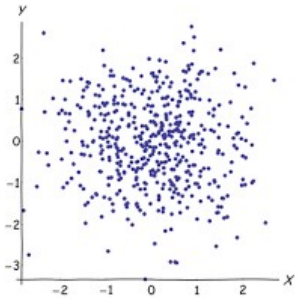
Data cleansing

- Realistically assume that data is **always** imperfect
  - Whether automatically collected or manually entered
    - Ex. Customer data, sensor data
- Imperfection is diverse
  - Contradicting samples
  - Missing data
  - Outliers/anomalies
  - Noise of all colors
- Consider the goal of the analysis
  - Anomaly detection? Classification? …
- Consider the percentage of imperfection
  - Small? You may decide to remove/ignore imperfect data
  - Large? You may decide on a default value. Choose one answer consistently, use a different algorithm, …
- Whatever you choose to do, e.g., discard records with missing values or fill in the missing values with the average, stick with it

Capture statistical assumptions resulting in feature selection/augmentation/reduction and cleansing

- Don’t remain at the descriptive statistic level
- Capture your statistical assumptions in a programmable and automated manner
  - Ex. Applying some statistical test
- Check at deployment and if you get raw data for training that the assumption still holds
- If it no longer holds, this is a red flag that may indicate
  - Your model needs to be reevaluated or
  - Learning needs to be repeated
- Example –
  - Missing data per feature is below 10%
  - Two features are correlated during training but not at deployment
  - Hands-on: Run pearson.py a few time to experiment with changes in linear correlation





Design experiments to find models and hyper-parameter values

- Reduces risk
  - Avoid magic numbers and create a robust solution that can be updated when the data changes
- A requirement for automating the learning from raw data
- Tune and compare ML performance metric of multiple models
- Define a search problem over a loss function that captures the tradeoffs
  - sklearn.model_selection.GridSearchCV
  - Ex. In a labeling hierarchy want the best accuracy and most leaf labels
  - Ex. Total over all inputs of instances with different label within same cluster and instances with same label in other clusters
- Ensemble methods based on bagging (generating new training sets using sampling with replacement). Ex., random forest, boosting -- part of the training set remains unused. For each classifier in the ensemble, a different part of the training set is left out. Can be used to estimate the generalization error and for model selection

Create a baseline to compare your model with

- Identify a naïve, potentially no-learning approach that is easy to implement
- Always compare your current version performance to that of your baseline
- Your model should be superior to the baseline
  - The baseline serves as a sanity check/reference model
  - Can also be utilized to mitigate the existence of limited labeled data
- Are there areas where it is not better?
  - Improve your model
  - Consider a hybrid approach
  - Is your baseline good enough?
  - Simplicity has advantages
- Ex. Regression test selection. We want to execute a subset of the tests that will find all the defects in the shortest time. We can simply select the tests that found the most defects in the past and run per past run time

ID	Defects	Duration
1	0	1h
1	1	10h
2	10	1h
…		

TESTING AND DEPLOYMENT

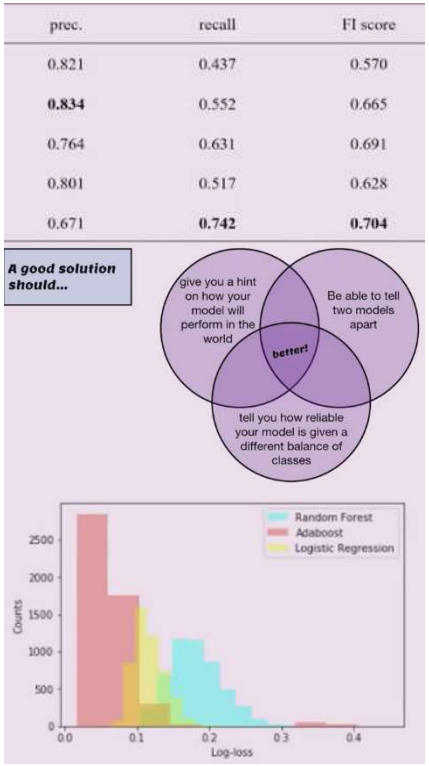
Provide confidence intervals rather than a single number – use non-parametric statistics and bootstrapping to test the system

Current best practice

- The test team obtains a ground truth test set
  - Per business process requirements
    - Otherwise, randomly chosen
  - Rule of thumb: 20% of the training set
    - This may be divided between:
    1. Development, used for learning, and
    2. Test, never used for learning
- Some standard loss function is used to obtain the average performance of the ML algorithm on the test set
  - Ex. For a classification task, accuracy of the machine learning model on the test set is obtained
  - System is expected to be stable over time (stationary)
  - Ideally, this is part of the acceptance criteria agreed with the client
    - Ex. F1 score at least 75%

Do not forget statistics

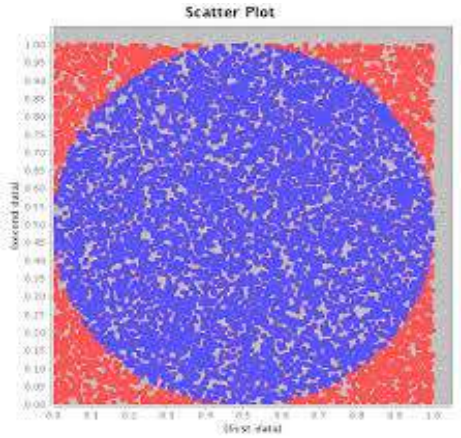
- Measure the ML performance metric on your test set but
  - Never use your test set for training
  - Or it will become a development set and you will need more data for a new test set
- Do not give a single average number, e.g., accuracy or F1. Instead, develop confidence intervals
  - An average is confusing and misleading
  - Ex. Giving a single number for processing multiple data formats. May be mis-performing on at least one.
  - Bootstrap over the test set multiple times, measure the ML performance metric, then compute the expected value and its confidence interval
- Understand the ML performance over projected field data
  - Anticipate and identify when field performance degenerates
  - Can project automatically or per the requirements (Ex. data formats)



Noah Dolev: 'Which is better? Telling any two things apart', PyData, Tel-Aviv, January 2018

"Easy"- improvements to the state of the practice

- Quantitative choice of the test set required size
  - Can be obtained using application of various versions of the law of large numbers
  - Many times required size is too large in practice
- The loss on the test set does not predict the real performance
  - Need to obtain a confidence interval instead of just an average estimate
  - This is independent of a confidence interval on the result of learning
  - Important also for setting expectations correctly
    - Client AND solution provider management
  - Ideally would use Monte Carlo and obtain repeated samples
  - Instead, bootstrapping on a single test set can be applied

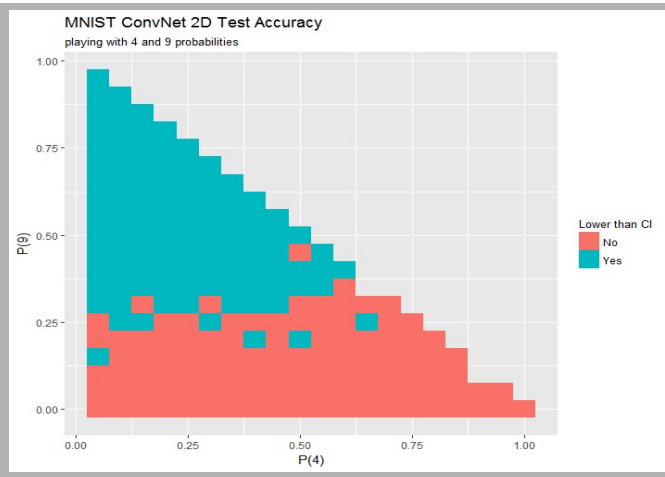


Obtain a confidence interval on the accuracy, A, using bootstrapping

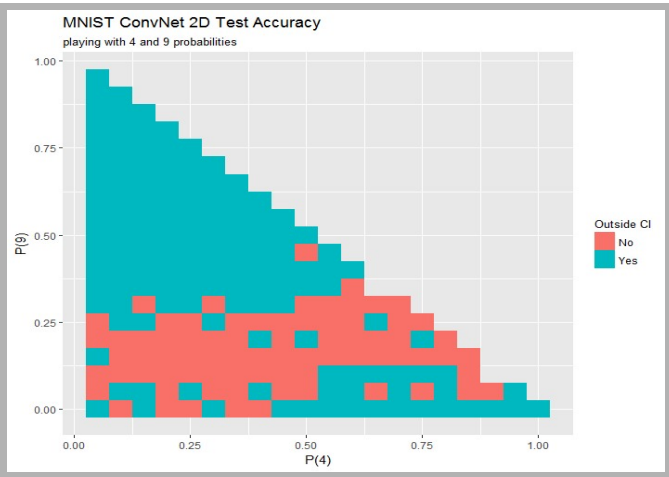
- Repeatedly sample X with replacements and obtain many estimates of A say A1, …AK
- Obtain the sampled average and standard deviation of A1,…,Ak
- By the central limit theorem A1,…Ak is distributed normal for large k
  - Obtain a confidence interval based on that
- Hands-on: Run regression.py, min.py, pearson.py a few times each and examine changes in confidence intervals

Identify the envelope of operation

- Are there regions ‘low’ enough around the accuracy ‘mountain’ that are no longer in the confidence interval (CI)?
- Bootstrap 95% confidence interval with 10K re-samples around the 99% accuracy of MNIST



Once the proportion of the digit 9 is higher than ~25% in your sample - expect to get worse results than your confidence interval indicated!



The confidence interval is true only for a very narrow region, which "keeps" the original distribution of about 10% for each digit

Size of the ground truth (setup)

- h is the ML model you have learned in some way
  - Your objective is now to project its performance
- h(x) = 1 - problem is coming from the program under test
- h(x) = 0 - problem is coming from the environment
- To test it, we chose a sample X - new samples
  - Or a test set that is randomly chosen from the original training set (BUT NOT USED FOR LEARNING)
  - We know the correct answer - f(x) for X

$$\mathbb{P}\left(\overline{X} - \mathbf{E}\left[\overline{X}\right] \geq t\right) \leq \exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

$$\mathbb{P}\left(\left|\overline{X} - \mathbf{E}\left[\overline{X}\right]\right| \geq t\right) \leq 2\exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

Obtaining the size of the ground truth using the Hoeffding inequality

- The statistical test - we count how many time f(x) = h(x) and divide it by |X| and get the average accuracy A
- We want to estimate Pr(h(x) = f(x)) in real LIFE (not on X).
- Desired claim - with high probability, the chance that A is far away from Pr(h(x) = f(x)) is small
- Need to determine how big X should be
- Answer – apply the Hoeffding inequality
  - Have the right side of the inequality less than some e > 0,
  - |X| >= -(ln t - ln 2)/2e^2. Choose t = e = 0.01.
    Then -(ln(0.01) - ln(2))/(2 * (0.01)^2) ~ 2300

$$\mathbb{P}\left(\overline{X} - \mathrm{E}\left[\overline{X}\right] \geq t\right) \leq \exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

$$\mathbb{P}\left(\left|\overline{X} - \mathrm{E}\left[\overline{X}\right]\right| \geq t\right) \leq 2\exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

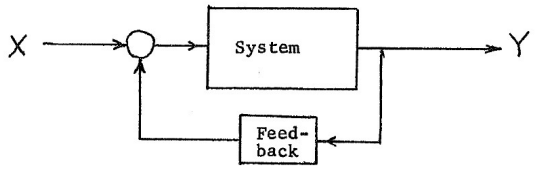
Estimate the probability of a labeling mistake and take it into account in testing (and development)

- Sample the entire training set to determine the percentage
  - Possibly per label
  - Obtain k data points
  - Each sampled data point is manually inspected to check if the label is correct or not
  - Obtain the empirical percentage of labeling error
    p = #(labeling_error)/k
  - Check that kp and k(1-p) is more than 5
  - Determine a confidence interval over p see https://onlinecourses.science.psu.edu/stat100/node/56 for details
- Use p and its obtained confidence interval to update your performance estimates
  - If I'm correct on the test set then it is only in probability (1-p)

$$\hat{p} \pm z^* \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

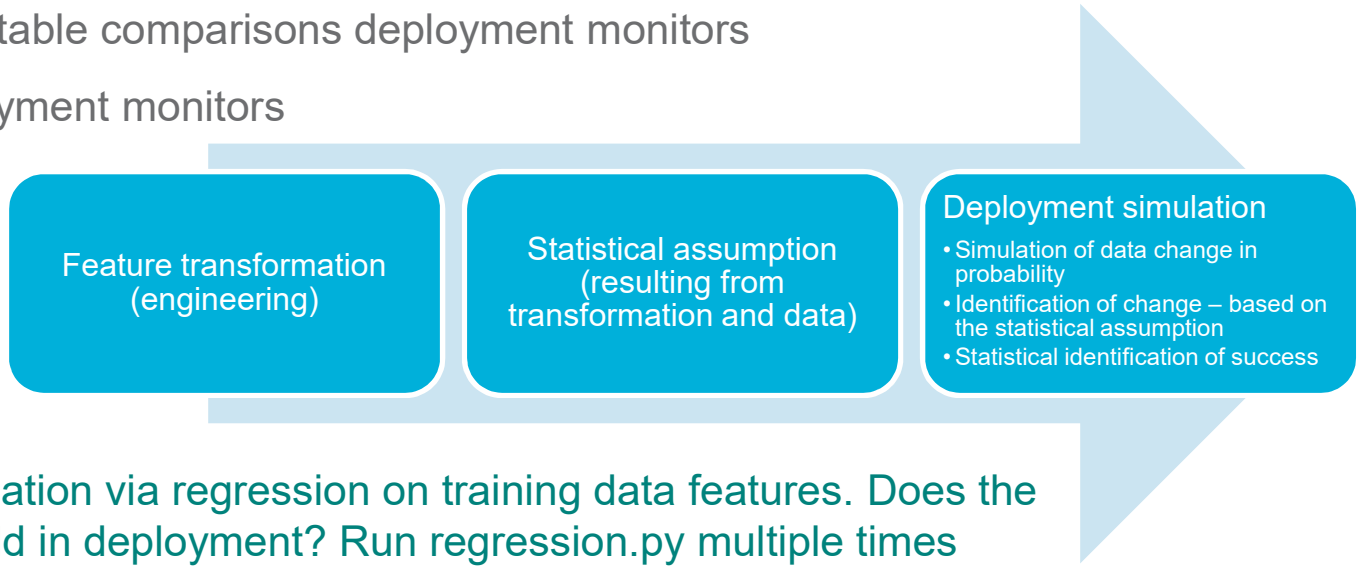
Implement a feedback mechanism for users to help fine tune the solution

- Is it possible to improve the labels or get more labeled data during deployment?
- This is only realistic if it is very easy for the user and is automatically collected and added
  - Ex. Clicks, page views, eye tracking,
- This is closely related to the relation between the business value metrics and the ML performance metrics see explicitly measure business value
- Often different users have different value metrics
- Ex. Specific cases over which the user expects perfection
  - Understand these and define as data bins to add to the testing



Introduce deployment monitors

- We talked about automatically extracting statistical properties that hold over the training data distribution and comparing them to the deployment data distribution throughout deployment
- We call these executable comparisons deployment monitors
- You can learn deployment monitors



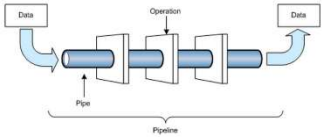
- Hands-on: Linear relation via regression on training data features. Does the same model (R^2) hold in deployment? Run regression.py multiple times

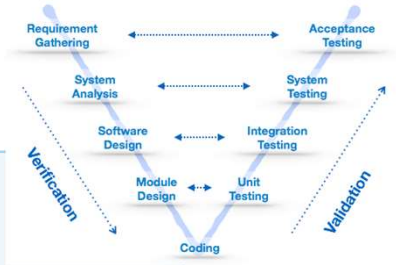
APPENDIX

Annotated reference

- Google's best practice paper on how to develop reliable ML solutions - https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/46555.pdf
- Andrew Ng, Machine Learning Yearning. Draft. 2018

What to do when?





ML test level

Component / Network	Requirements	Design	Development	Testing
Network	More work needed here How are the ML components connected (a pipe line? Independent?) What should be the ML performance metric value for each? Identify input, map output to business value			
Component	Apply statistics Business value	Apply statistics Use all relevant data Data of valuable volume Relation between statistical assumptions and logical contracts Feedback mechanism	Apply statistics Get to a supervised learning problem Convex optimization problem Experiments for parameters Automate pipe line Capture statistical assumptions	Apply statistics Business value Test on the raw data
	Requirements Loss function Optimization objective Business objective ML performance metric	**Design** Data pipeline Feature extraction	**Development** Learning	**Testing** Testing on test set (generally time based)

Lifecycle of ML model

Workshop abstract

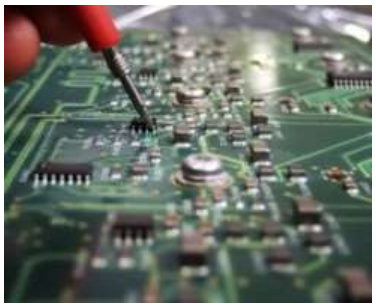
There is often a gap between the Machine Learning (ML) model performance in training, which is acceptable, and the model performance in deployment, which drifts away from acceptable to unacceptable performance.

System quality is decided by the combination of data quality, user/business requirements and suitability of the ML models. In contrast to traditional applications, ML testing is statistical at all levels. Today, there is a lack of methodology and technology for supporting ML testing.

This workshop introduces non-parametric statistical control into the process of ML solution development. It does so through a set of heuristics for creating a successful ML solution accompanied by examples and code utilities

Redefine your problem as a convex optimization problem and use off-the-shelf convex optimization software packages

- TBC

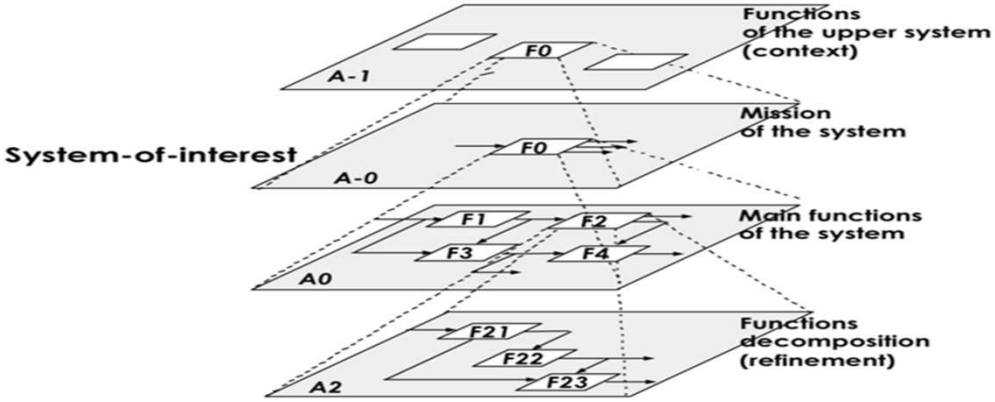


Reliable cognitive solutions

Eitan Farchi, Orna Raz

The problem

- Traditional systems are built through decomposition and the recursive definition of what the system should do and how it is going to do it
  - This also applies to cognitive systems but now there are components that are built
    - using machine learning and
    - are dependent on sampled data and its manipulation



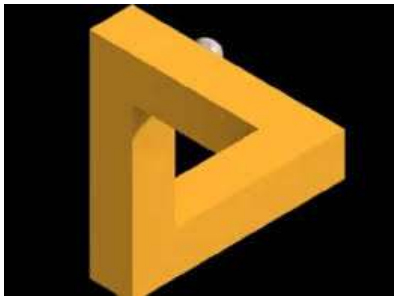
The problem (continued)

- As a result new reliability concerns arise:
  - How good is the marriage between the business requirements, loss function, and relevant/non relevant data?
  - Assumptions on data
    - Is the sampled data representative at learning time?
    - Over time did the production data distribution change?
    - These questions apply to data transformations as well
  - Does the architecture best utilize ML capabilities given the available data?
  - What part of the learning is manual/artistic and might not converge without human intervention?
  - Are the learned models readable/debug-able/verify-able?
  - How to define the system acceptance test for a system that is correct only part of the time by design?
  - Composition of ML components as they are designed to be correct in probability

- Traditional software correctness becomes a necessary condition and “takes a back seat” ☺

Design time - marry business requirements with relevant data that can be successfully learned through appropriate optimization objectives

- Iterate to create a magic triangle
- Discuss the business requirement
- Define and review possible optimization objectives/loss functions
  - Can the objective function be kept concave to obtain an easy search problem?
  - Can learning bias be sufficiently introduced in the model
- Is the data
  - Labeled?
  - Biased?
  - Relevant?
  - On the right order wrt number of features and number of samples?
  - Expected to be stable over time (changes or cycles)?



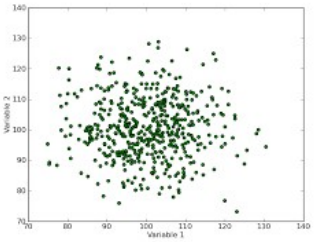
Design time – Can/should machine learning be applied?

- Do I have enough data?
  - < 100 samples, will not accurately be able to build a model
- Is the data biased? What is the distribution of the data?
  - If the data is skewed, you will have a skewed model
  - i.e. 99.9% of all network packets are non-malicious, then we need to use anomaly detection rather than classification
  - Classification bin splits matter
  - i.e. 90% of all defects are valid, 10% are user error then we need to apply text analysis
- Is the data labeled?
  - If not, restricted to unsupervised learning
  - Accurate labeling is an expensive but valuable endeavor
- Let the data tell you what is important
  - Sometimes, as humans, we cannot detect hidden or subtle patterns
  - If the feature set is too large, by data size or computation time, attempt to apply dimensionality reduction or more strict feature selection
- Model accuracy matters
  - < 50%, might as well flip a coin
  - < data distribution, might as well choose largest bin
  - i.e. If valid defects occur 70% of the time, my model needs to be better than 70% accurate or one should just always pick valid as a prediction
- Sometimes, need to just experiment with data

Development – the data pipeline

- The data pipeline typically contains data manipulations of all sorts including feature extraction, selection, aggregation, abstraction, anonymization, etc.
  - Pipeline is complex containing a lot of steps that might cause loss of information, and be incorrect now or in the future
- Make the data pipeline automated, traceable, self verifying, and compliant
- Explicitly identify assumptions related to data manipulation
  - Translate these assumptions into checks
  - Apply them whenever relearning is conducted
  - Apply them at production time to identify drifts that will hinder system correctness

- High correlation between variables lets you remove one
  - Test for correlation at production time shows that the two are not correlated
- Principle component analysis was used to reduce the number of features
  - That introduced an implicit mistake as the system must have a readable model and the eigenvectors were not
- The learning model estimates a conditional probability P(A|B) assumed to be of a Bernoulli distribution
  - The training set estimation was significant the first time around but at relearning time data for P(B|A) is no longer significant
- Yet more fundamental - your self driving car is driving a France highway but learned using autobahn (German) data



Change is the only constant

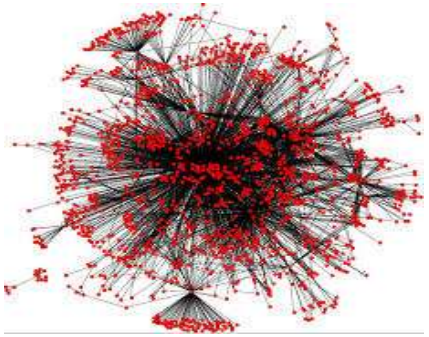
- Assume production data will be different over time
  - The factors that determine if a test failure is indicating a test problem or a program under test problem changes over the life cycle of a project
- Analyze production data
  - For early identification of breakage in learning assumptions
  - To determine when relearning is needed
  - To determine the window of time to be used for learning

Development - learning

- Prefer less accurate learning that is
  - not artistic
  - automated
  - repeatable
- Incorporate learning bias in the choice of the model (and features ☺)
- Automate the choice of hyper parameters such as the rate in gradient descent
- Compare the stability of different alternative solutions
- Spend time and effort to make the chosen model readable

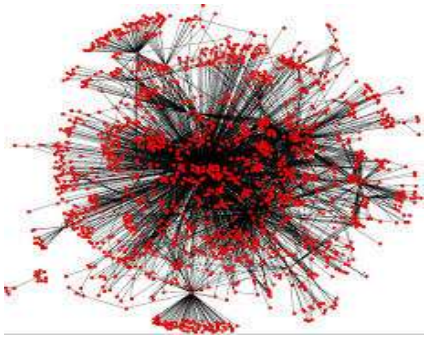
Black box acceptance test

- Independently create tests from the business requirements
  - They should be derived independently than the machine learning sub components loss functions
    - No accident is the business requirement but machine learning sub-components may have many other formulated objectives that lead to that ultimate goal
  - Some of them may be statistic in nature
    - Treat as a statistic experiment
    - Use combinatorial test design to handle exponential test spaces
  - If we had this model last year…
    - Train over the data from years 1-4
    - Test with data from year 5



White box acceptance test

- Consider the system as a white box
  - Review and identify all assumptions made during the learning process
  - Test production data for the assumption
  - It may look silly but it is not….☺



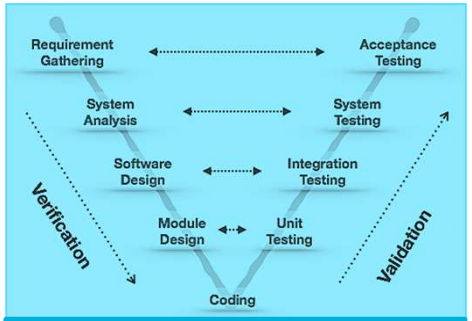
Check list/maturity self assessment

- ❑ Data transformation assumptions are identified
- ❑ ML hyper parameters and other decision effecting learning are identified
- ❑ Automatic tests are in place to check for data transformation assumption when learning
- ❑ ML stage two above is automated
  E.g. linear search is applied for gradient descent
- ❑ Automated black box acceptance test is implemented
- ❑ Automated white box acceptance test is implemented
  - ❑ i.e., applying 3 above on production data

Aligning process with testing stages for ML solutions

Process Stages

- Requirements
- Design
- Development
- Testing
- Deployment



Data, ML, Statistical Analysis

Testing Stages

- Unit testing
- Integration testing
- System monitoring