

Roadmap for Machine Learning

Discussion about **A**rtificial **I**ntelligence; **M**achine **L**earning; **D**eep **L**earning and **D**ata **S**cience
Made by Daniel Bourke (any questions, feel free to ask). Copy/use whatever you need.

1.0 Machine Learning Roadmap (Objectives)

- [1] Machine Learning **PROBLEMS**
- [2] Machine Learning **PROCESS** (steps in a machine learning project)
- [3] Machine Learning **TOOLS** (tools you can use to get the job done)
- [4] Machine Learning **MATHEMATICS** (what's running under the hood when you write machine learning code)
- [5] Machine Learning **RESOURCES** (how to learn, places to visit, etc)

1.1 ML problems

☐ CATEGORIES type Problems (types of learning):

- i). **SUPERVISED Learning:** You have **data** and **labels**. The model tries to learn the *relationship between data and labels*.
 - ☹ For example, you have 10,000 photos of **cats** and **dogs** (5,000 each) and the labels for which **photo** contains which **animal** (photo 1 = dog, photo 2 = cat).
 - ☹ It works for numbers too. EG: 10,000 **houses** and their **selling price**. Use the information about the **houses** to try and **predict** the **selling price**.
- ii). **UNSUPERVISED Learning:** You have **data** but **NO labels**, the model tries to find patterns in data without something to reference on.
 - ☹ For example, you've got 50,000 **transactions** and 49,997 of them are similar but 3 of them are completely outlandish, -which 3? This kind of problem is called **anomaly detection**. Other common problems include clustering and dimensionality reduction (PCA).
- iii). **REINFORCEMENT Learning:** An **agent** (algorithm) performs actions in an **environment** and is **rewarded** or **penalized** based on the whether the actions were *favorable or not*. The strategy an agent learns (based on the actions it gets rewarded for) is called a **policy**. (Mix of **Labeled** and **Un-Labeled** data).
 - ☹ For example, the agent could be a **ChessPlayerX** (a chess playing algorithm), the environment is a virtual chessboard and the actions are moving pieces. If a piece gets moved to a negative position, the algorithm gets penalized, if it takes an opponent's piece it gets rewarded.
- iv). **TRANSFER learning (Saves time):** Take knowledge from one model and use it in your own.
 - ☹ Train a model (collecting and processing data) is time consuming. Eg: **TESLA** trained car took **8 years**.
 - ☹ For example, take all of the text from Wikipedia, learn the relationships between words and use these underlying relationships to help you build your insurance quote classifier.

☐ PROBLEM DOMAIN

1. **Classification**
2. **Regression**
3. **Sequence to sequence** (seq2seq)
4. **Clustering**
5. **Dimensionality Reduction**

[1] **Classification**

- i). **Binary classification:** Is this **email spam or not spam**? Is this a **photo of a cat or a dog**?
- ii). **Multi-class classification:** Is the **traffic light** green, yellow or red? What **breed of dog** is in this photo?
- iii). **Multi-label classification:** What **items** does this **photo** contain? What **topics** is this YouTube **video** about?

▶ **Evaluation metrics:**

- ➡ **Confusion matrix:** **True positives and negatives** and **false positive and negatives** plotted against each other. Useful for seeing where a **model** is **getting confused**. E.g. How often it predicts the right and wrong classes.

- ⇒ **Accuracy:**
 - **Blog:** Out of 100 examples, how many will I get right?
 - **Note:** Unreliable when number of samples differ. E.g. 9999 examples of class A and 1 example of class B, a model which predicts all class A is 99.999% accurate. In this case, use precision and recall instead.
- ⇒ **f1 score (combination of precision and recall):**
 - **Precision:** Accuracy of the positive predictions. Of the positive predictions, what proportion are correct? True Positives/(True Positives + False Positives). A model with a precision of 1.0 has no false positives.
 - **Recall:** Ratio of true positive predictions, also known as sensitivity. True Positives/(True Positives + False Negatives). A model with a recall of 1.0 has no false negatives.
 - **Precision/Recall tradeoff:** Increasing precision reduces recall and vice versa. If you choose to optimize for one or the other, you'll have to decide whether less false negatives (higher recall) or less false positives (higher precision) are better. For example, if your predicting the presence of cancer in an X-ray, is it better to get false positives or false negatives? If anyone says "We need 95%+ recall", ask, "at what precision?".
 - **Precision/Recall curve:** Plot the precision versus recall at different classification thresholds. Allows you to choose a precision value at a given recall value (as one increases, the other decreases).
- ⇒ **ROC Curve/Area Under Curve (AUC):** The receiver operating characteristic (ROC) curve is a common way to evaluate binary classifiers. It plots the false positive rate against the true positive rate (recall). The area under the curve (AUC) is the space underneath the ROC curve. A perfect classifier will have an AUC of 1.0 (you can use this value to compare different classifiers).
 - **Note:** Generally, if you don't have many positive examples or if you care more about false positives than false negatives, you should prefer the precision/recall curve. And if not, use the ROC/AUC.

[2] **Regression:** Example of problems:

- Given the number of bedrooms, number of bathrooms and house location, predict the sale price of a given house.
- How much will Bitcoin be worth tomorrow?
- ▶ **R^2 (r-squared):** How well does my model fit? A perfect model scores 1.0, a model predicting the mean of every label gets 0.
- ▶ **MSE (mean square error):** Makes outliers stand out more. Use if being 10% off is more than twice as bad as being 5% off.
- ▶ **MAE (mean absolute error):** All errors are on the same scale. E.g. if trying to predict 100, predicting 99 is the same error as predicting 101.

[3] **Sequence to sequence (seq2seq):** Example of problems

- A Given a sequence of English text, translate it into French.
- Build a device which responds to speech commands (Alexa). Speech goes in (as a sequence of vibrations in the air), gets converted to text and interpreted as a command. Then another sequence (in the form of a generated voice) is played back.

[4] **Clustering:** Example problems:

- Group the customers of the business by their purchases. Labels for the groups aren't given (making it unsupervised problem). Once groups are found, labels can be added.

[5] **Dimensionality Reduction:** Example problems:

- Reduce the number of inputs into a model by mapping the features into lower dimensional space. For example, say you have 100 inputs (which may be too many for your model), how can you reduce this down to the 10 most important inputs? One technique for doing so is principal component analysis (PCA).
- Representation learning/feature learning (note: this may be better off on its own branch). Learn the underlying features of a collection of data for use in a downstream task. For example, have an algorithm read the entirety of Wikipedia to form a representation of how words should co appear in the context of others (build a language model). How do you choose (or learn) the best features/representation of a data a machine learning model can learn on?

1.2 ML process

There are a lot of steps here. But don't treat this as an exhaustive list. I've only added steps for the most-common steps you'll come across or ones I've had direct hands-on experience with.

👉 **NOTE:** *The number one thing to remember is:* machine learning is experimental. Try something, see if it works, if not, try something else. Your primary goal for machine learning problems should be reducing the time between your experiments.



Steps in a Machine learning Project:

1. Data COLLECTION
2. Data PREPARATION

3. TRAIN model on data
 - a. Choose an **algorithm**,
 - b. Overfit the **model**,
 - c. Reduce overfitting with **regularization**
4. ANALYSIS/Evaluation
5. SERVE model (deploying a model)
6. RETRAIN model

1.2.1 **Data COLLECTION:**

❑ **Question to ask:**

- 🔊 What *kind of problem* are we trying to solve? (see machine learning problems)
- 🔊 What *data sources* already exist?
- 🔊 What *privacy* concerns are there?
- 🔊 Is the data *public*?
- 🔊 *Where* should we *store* the data?

❑ **Types of data:**

- ➡ **Structured data:** data which appears in tabulated format (rows and columns style, like what you'd find in an Excel spreadsheet). It can contain different types of data, for example, **numerical, categorical, time series**.
 - i. **Nominal/categorical:** One thing or another (mutually exclusive). For example, for **car sales**, **color** is a category. A car may be **blue** but not **white**. Order does not matter.
 - ii. **Numerical:** Any continuous value where the difference between them — matters. For example, when selling houses. \$107,850 is more than \$56,400.
 - iii. **Ordinal:** Data which has order but the distance between values is unknown. For example, questions such as. how would you rate your health from 1-5? 1 being poor, 5 being healthy. You can answer 1, 2, 3, 4 or 5 but the distance between each value doesn't necessarily mean an answer of 5 is 5 times as good as an answer of 1.
 - iv. **Time series:** Data across time. For example, the historical **sale values** of **Bulldozers** from **2012-2018**.
- ➡ **Unstructured data:** Data with no rigid structure (**images, video**, natural language **text, speech**).

1.2.2 **Data PREPARATION**

❑ **Exploratory data analysis (EDA)** (learning about the data you're working with):

- ➡ What are the **feature variables (INPUT)** and the **target variables (OUTPUT)**? For example, for predicting heart disease, the **feature** variables may be a person's **age, weight**, average **heart rate** and their level of physical activity. And the **target** variable will be whether or not they have **heart disease** (note: this example has been very simplified).
- ➡ What **kind of data** do you have? **Structured, Unstructured, Categorical, Numerical**.
 - ❖ Create a data dictionary for what each feature is. For example if you've got a column of numbers called "hr", how would someone else know that actually means Heart Rate?
- ➡ Are there **missing values**? Should you remove them or fill them with feature imputation (see below)?
- ➡ Where are the **outliers**? How many of them are there? Are they out by much (3+ **standard deviations**)? Why are they there?
- ➡ Are there questions you could ask a **domain expert** about the data? For example, would a heart disease physician be able to shed some light on your heart disease **dataset**?

❑ **Data preprocessing (preparing your data to be modeled):**

- ➡ **Feature imputation (filling missing values):** A machine learning model can't learn on data that isn't there.
 - i). **Single imputation:** Fill with **mean, median** of column.
 - ii). **Multiple imputation:** Model other missing values and fill with what your model finds.
 - iii). **KNN (k-nearest neighbours):** Fill data with a value from another *example which is similar*.
 - iv). **Many more**, such as, random imputation, last observation carried forward (for time series), moving window, most frequent.
- ➡ **Feature encoding (turning values into numbers):** A machine learning model requires all values to be numerical.
 - 1) **OneHotEncoding:** Turn all unique values into lists of 0's and 1 's where the target value is 1 and the rest are 0's. For example, with car colors green, red, blue, a green car's color feature would be represented as [1,0, 0] and a red one would be [0,1,0].
 - 2) **LabelEncoder:** Turn labels into distinct numerical values. For example, if your target variables are different animals, such as dog, cat, bird, these could become 0, 1,2 respectively.

- 3) **Embedding encoding:** Learn a representation amongst all the different data points. For example, a language model is a representation of how different words relate to each other. Embeddings are also becoming more widely available for *structured (tabular) data*.

➡ **Feature normalization (scaling or standardization):** When your numerical variables are on **different scales** (e.g. **number_of_bathrooms** is between 1 and 5 and **size_of_land** between 5,000 and 20,000 sq feet), some machine learning algorithms don't perform very well. Scaling and standardization help to fix this.

- i. **Feature scaling** (also called **normalization**): Shifts your values so they always appear between **0** and **1** (similar to **Fuzzification**). This is done by subtracting the **min** value and dividing by the **max** minus the **min**. For example, 1 and 6 bathrooms would become something like 0.2 and 0.8 respectively (between 0 and 1 but the numerical relationship is still there).
- ii. **Feature standardization:** Standardizes all values so they have a **mean** of **0** and **unit** variance. It happens by subtracting the **mean** and dividing by the **standard deviations** of that particular feature. Doing this means *values do not end up between 0 and 1* (their range is uncapped). Standardization is more robust to outliers than feature scaling.

➡ **Feature engineering:** Transform data into (potentially) more *meaningful representations* by adding in **domain knowledge**.

- ⇒ **Decompose**, for example, turn a date (such as 2020-06-18 20:16:26) into **hour_of_day**, **day_of_week**, **day_of_month**, **is_holiday**, etc.
- ⇒ **Discretization (turning larger groups into smaller groups)**. Data **discretization** refers to a method of converting a huge number of data values into smaller ones so that the evaluation and management of data become easy.
 - ❖ For **numerical variables**, let's say **age**, you may want to move it into buckets, such as, **over_50** or **under_50**. Or, **21-30**, **31-60**. etc. This process is also known as **binning** (putting data into different **bins**).
 - ❖ For **categorical variables** such as car **color**, this may mean combining colors such as **light-green**; **dark-green**; and **lime-green**; into a single **green** color.
- ⇒ **Crossing and interaction features:** In other words, *combining two or more features*. Difference between two features, such as using **house_last_sold** and **current_date** to get **time_on_market**.
- ⇒ **Indicator features:** Using other parts of the data to **indicate** something **potentially significant**.
 - ❖ Create an **X_is_missing** feature for wherever a column **X** contains a **missing value**.
 - ❖ If you're analyzing traffic sources from the web, you could add **is_paid_traffic** as feature for advertisements which have been paid for.
 - ❖ Does a particular sample satisfy more than 1 criteria? Such as if you were analyzing car sales data, does the car have less than 100,000 KM's, is automatic and under 10-years old? Perhaps you know from experience these cars are generally worth more. You could make a special feature called **under_100km_auto_under_10yo**.

➡ **Feature selection:** selecting the most valuable features of your dataset to model. Potentially reducing overfitting and training time (less overall data and less redundant data to train on) and improving accuracy.

- ⇒ **Dimensionality reduction:** A common dimensionality reduction method. **PCA** or **Principal Component Analysis** takes a larger number of **dimensions (features)** and uses **linear algebra** to **reduce** them to **less dimensions**. For example, say you have **10 numerical features**, you could run **PCA** to reduce it down to **3**.
- ⇒ **Feature importance (post modelling):** Fit a model to a set of data, then inspect which features were most important to the results, remove the least important ones.
- ⇒ **Wrapper methods:** Such as **Genetic Algorithms** and **Recursive Feature Elimination** involve creating large subsets of feature options and then **removing** the ones which **don't matter**. These have the potential to find a **great set of features** but can also require a **large** amount of **computation time**. **TPot** does this.

➡ **Dealing with imbalances:** does your data have 10,000 examples of one class but only 100 examples of another?

- ⇒ **Collect more data** (if you can).
- ⇒ **Use the scikit-learn-contrib imbalanced-learn package** (a scikit-learn compatible Python library for dealing with imbalanced datasets).
- ⇒ **Use SMOTE** (synthetic minority over-sampling technique) which creates synthetic samples of your minor class to try and level the playing field. You can access an implementation of SMOTE contained within the scikit-learn-contrib imbalanced-learn package.
- ⇒ A helpful (and technical) paper to look at is **'Learning from Imbalanced Data'**. You could use this as a starting point to go onto the next thing.

❑ **Data Splitting:**

- ➡ **Training set (usually 70-80% of data):** Model learns on this.
- ➡ **Validation set (typically 10-15% of data):** Model **hyper-parameters** are tuned on this.
- ➡ **Test set (usually 10-15%):** Models final performance is evaluated on this. If you've done it right, hopefully the results on the test set give a good indication of **how the model should perform in the real world**. Don't use this dataset to tune the model.

1.2.3 **TRAIN model on data** (3 steps: **CHOOSE** an **algorithm**, **OVERFIT** the **model**, **REDUCE** overfitting with **regularization**):

1. Choosing an ALGORITHM
 - a. Supervised Algorithm
 - b. Unsupervised Algorithm
2. Type of learning
 - a. Batch learning
 - b. Online learning
 - c. Transfer learning
 - d. Active learning
 - e. Ensembling
3. Underfitting
4. Overfitting
5. Hyperparameter Tuning

□ **Choosing an ALGORITHM**

i. **SUPERVISED** Algorithm

ii. **UNSUPERVISED** Algorithm

i. **SUPERVISED Algorithm:**

- ➡ **Linear Regression:** Draw a **line** that best fits **data scattered** on a **graph**. Produces continuous variables (e.g. height in inches).
- ➡ **Logistic Regression:** Predicts a **binary outcome** based on a series of **independent variables**. For example, if you're trying to **predict** whether someone has **heart disease** or not based on their **health parameters**.
- ➡ **k-Nearest Neighbors:** Find the '**k**' **examples** which are **most similar** to each other. Then, given a new sample, which is the **new sample** most closely aligned with?
- ➡ **Support Vector Machines (SVMs):** Can be used for **Classification** or **Regression**. Try to find best way to **separate data points** using **multiple planes** (referred to as **Hyperplanes**).
Imagine a road running through a city with red cars on the left and green cars on the right, the SVM is the road. With more data, you might need more dimensions (more roads on different angles and cars on different heights).
- ➡ **Decision Trees and Random Forests:** Can be used for **Classification** and **Regression** (valuable for **structured data**).
 - **DECISION TREES** split data based on criteria such as, "**is over 50**" and "**has average heart rate lower than 65**" eventually getting to a point where the data can't be split anymore (you can define this).
 - **RANDOM FORESTS** are a **combination** of many **decision trees**, effectively **leveraging and combining the choices of many Models** (this technique of using a **combination of models** is known as **ensembling**).

👉 See a **great way to visualize decision trees** by **explained.ai**
- ➡ **AdaBoost/Gradient Boosting Machines (also just known as boosting):** Can be used for **classification** or **regression**. Asks the question, can a **series of weak learners** be turned into a **strong learner**? For example, **train a series of weak learners** whose job it is to **improve each other** (another example of an **ensemble method**, combining multiple weaker models to create a better one).
 - ✓ **XGBoost** algorithm
 - ✓ **CatBoost** algorithm
 - ✓ **LightGBM** algorithm
- ➡ **Neural networks (also called deep learning):** Can be used for **classification** or **regression**. Takes a series of inputs, manipulates the inputs with **linear** (dot product between weights and inputs) and **nonlinear functions** (activation function). Do this a multiple of times (at least once for each neuron in a model).

👉 The importance of **linear** and **non-linear functions** (straight and non-straight lines) means neural networks can use this combination to estimate almost anything.

 - ✓ **Convolutional** neural networks (typically used for **computer vision**).
 - ✓ **Recurrent** neural networks (typically used for **sequence modeling**).
 - ✓ **Transformer** networks (can be used for **vision** and **text**, starting to replace **RNNs**).

ii. **UNSUPERVISED Algorithm:**

- ➡ **Clustering:** For example, K-Means clustering. Choose **k'** number of clusters, each cluster receives a **centre node** (called a **centroid**) at random and with each iteration the **centre** nodes attempt to **move farther** away from each other. Once the **centroids stop moving**, each sample is assigned a value equivalent to the **closest centroid**.

➡ Visualization and DIMENSIONALITY reduction:


- ➡ **Principal Component Analysis (PCA):** Reduce data from **higher dimensions** to **lower dimensions** whilst attempting to preserve the variance.
- ➡ **Auto-encoders:** Learn a **lower dimensional encoding** of data. For example, **compress an image** of 100 pixels into 50 pixels representing (roughly) the same information as the 100 pixels.
- ➡ **t-Distributed Stochastic Neighbor Embedding (t-SNE):** Good for visualizing **higher-dimensional** data in a **2D** or **3D** space.


➡ Anomaly detection:


- ➡ **Auto-encoder:** Use an auto-encoder to **reduce the dimensionality** of the inputs of a system and then try to **recreate** those **inputs** within some threshold. If the **recreations aren't able to match the threshold**, there could be some sort of outlier.
- ➡ **One-class classification:** Train a model on **only one-class** (e.g. normal events of **computer network traffic**, which are usually in abundance), if anything **lays outside** of this class, it may be an **anomaly**. Algorithms for doing so include, **one-class K-Means**, **one-class SVM** (support vector machine), **isolation forest** and **local outlier factor**.


□ Type of learning:

- Batch learning:** All of your **data** exists in a big **static warehouse** and you **train** a model **on it**. You may train a new model once per month once you get new data. Learning may take a while and isn't done often ("don't stuff this one up"). Runs in production without learning (though can be **retrained** later).
- Online learning:** Your **data** is **constantly** being **updated** and you **constantly train** new models on it. Each learning step is usually **fast** and **cheap** (as opposed to batch learning). Runs in production and **learns continuously**.
- Transfer learning:** Take the **knowledge** one model has learned and use it — with your own. Gives you the ability to leverage **SOTA** (state of the art) models for your own problems.
 - ➡ Helpful if you **don't have much data** or **vast compute resources**. Use resources such as **TensorFlow Hub** or **PyTorch Hub** for broad model options and **HuggingFace transformers** (NLP models) and **Detectron2** (computer vision models) for specific models.

 TensorFlow Hub

 PyTorch Hub

 HuggingFace transformers


 Detectron2
- Active learning:** Also referred to as "**human in the loop**" learning. A **human expert interacts** with a model and provides updates to labels for samples which the model is most uncertain about.
 - ➡ Example of how **Nvidia** uses active learning for their **self-driving car** models.
- Ensembling:** Not really a form of learning, more **combining algorithms** which have **already learned** in some way to get better results. For example, leveraging the "wisdom of the crowd".


□ **Underfitting:** Happens when your model doesn't perform as well as you'd like on your data. Try **training** for **longer** or a more advanced model.

□ **Overfitting:** Happens when your **validation loss** (how your model is performing on the validation dataset, **lower is better**) starts to increase. Or if you don't have a validation set, happens **when the model performs far better on the training set than on the test set** (e.g. 99% accuracy on training set, 67% accuracy on the test set). Fix through various **regularization techniques**.


☞ **Regularization:** a collection of techniques to **prevent/reduce Overfitting**.

- L1 (lasso) and L2 (ridge) regularization:** **L1 regularization** sets **unneeded feature coefficients** to **0** (performs feature selection on **which features are most essential** and which aren't, useful for model **explainability**). **L2** constrains a models features (won't set them to 0).
- Dropout:** **randomly remove parts** of your model so the rest of it has to become **better**.
- Early stopping:** Stop your model **from training** before the **validation loss** starts to increase too much or more generally, any other **metric** has **stopped improving**. Early stopping usually implemented in the form of a model callback.
- Data augmentation:** **manipulate** your **dataset** in **artificial** ways to make it harder to learn. For example, if you're dealing with **images**, randomly rotate, skew, flip and adjust the height of your images. This makes your model have to **learn similar patterns across different styles** of the same image (harder). Note: since this can be compute intensive, it's a good idea to do this in memory. See a functions like **ImageDataGenerator** in **Keras** or transforms in **torchvision**.

 **Image Transformation in PyTorch**

 **Image Augmentation in Keras/TensorFlow**

 **EDA** (easy data augmentation): **Text Augmentation** with **Python** (boost performance on text classification tasks)

 **TextAttack:** framework for adversarial attacks, data augmentation and modeling training in **NLP**.

- v. **Batch normalization:** standardize inputs (zero mean and normalize) as well as adding two parameters (beta, how much to offset the parameters for each layer and epsilon to avoid division by zero) before they go into the next layer. This often results in faster training speeds since the optimizer has less parameters to update. May be a replacement dropout in some networks.

□ **Hyper-parameter Tuning** (run a bunch of experiments with different model settings and see which works best):

- ☞ **Setting a learning rate:** (often the most important hyper-parameter), generally, high learning rate = algorithm rapidly adapts to new data, low learning rate algorithm adapts slower to new data (e.g. for transfer learning).
- ☞ **Finding the optimal learning rate:** Train the model for a few hundred iterations starting with a very low learning rate (e.g. $10e-6$) and slowly increase it to a very large value (e.g. 1). Then plot the loss versus the learning rate (using a log scale for learning rate), you should see a U-shaped curve, the optimal learning is about 1-2 notches to the left of the bottom of the U-curve.
- ☞ **Learning rate scheduling** (use Adam optimizer) involves decreasing the learning rate slowly as model learns more (gets closer to convergence).
- ☞ **Cyclic learning rate:** Dynamically change the learning rate up and down between some threshold and potentially speed up training.

☞ See: p326 **Hands-on Machine Learning BOOK** Edition 2.

☞ **[Very useful paper]** A disciplined approach to neural network hyperparameters by Leslie Smith. Covers learning rate, batch size, momentum, weight decay and more.

☞ Other hyperparameters you can tune:

- ✓ **Number of layers** (deep learning networks)
- ✓ **Batch size** (how many examples of data your model sees at once). Use the largest batch size you can fit in your GPU memory. If in doubt, use batch size 32 (see Yann LeCunn's Tweet, and the paper attached of course).
- ✓ **Number of trees** (decision tree algorithms)
- ✓ **Number of iterations** (how many times the model goes through the data). Note: Instead of tuning iterations, use early-stopping instead.
- ✓ Many more... depends on the algorithm you're using. Search "[algorithm_name] hyperparameter tuning".

1.2.4 Analysis/Evaluation

□ **Evaluation metrics**

[1] **Classification:**

- i. Accuracy
- ii. Precision
- iii. Recall
- iv. f1
- v. Confusion matrix
- vi. Mean average precision (object detection)

[2] **Regression:**

- i. MSE (mean squared error)
- ii. MAE (mean absolute error)
- iii. R^2 (r-squared)

[3] **Task-based metric** (create one based on your specific problem). For example, for self-driving cars, you might want to know number of disengagements".

□ **Feature importance:** Which features contributed most to the model? Should some be removed? Useful for model explainability, for example, telling someone, "the number of bedrooms is most important when predicting a house price".

□ **Training/inference time/cost:**

- ☞ How long does a model take to train? Is this feasible?
- ☞ How long does inference take? Is it suitable for production?

□ **What-if tool:** how does my model compare to other models? What if I changed something in the data? How does this effect the outcome?

□ **Least confident examples:** what does the model get wrong? (usually the long tail, instances you don't have many examples of)

□ **Bias/variance trade-off:** High bias results in underfitting and a lack of generalization to new samples, high variance results in overfitting due to the model finding patterns in the data which is actually random noise.

1.2.5 Serve model (deploying a model)

- ❑ Put the model into production and see how it goes. Evaluation metrics in vivo (in a notebook) are great but until it's in production, you won't know how it performs for real.
- ❑ Tools you can use:
 - ☞ **TensorFlow Serving** (part of TFX, TensorFlow Extended)
 - ☞ **PyTorch Serving** (TorchServe)
 - ☞ **Google AI Platform**: make your model available as a REST API
 - ☞ **Sagemaker**
- ❑ **MLOps**: Where **Software Engineering** meets **Machine Learning**, essentially all the technology required around a machine learning model to have it working in production.

1.2.6 Retrain model

- ❑ See how the model **performs after serving** (or **prior** to serving) based on various **evaluation metrics** and revisit the above steps as required (remember, machine learning is very experimental, so this is where you'll want to track your data and experiments).
- ❑ You'll also find your **models predictions** start to **'age'** (usually not in a fine-wine style) or **'drift'**, as in when data sources change or upgrade (new hardware, etc). This is when you'll want to **retrain** it.

1.3 Machine Learning TOOLS

- ❑ **The TOOLBOX**
 - [1] **Pretrained models (for transfer learning):**
 - ⇒ **TensorFlow** Hub
 - ⇒ **PyTorch** Hub
 - ⇒ Natural language processing: **HuggingFace** Transformers
 - ⇒ Computer vision: **Detectron2**
 - [2] **Experiment tracking:**
 - ⇒ **TensorBoard**: Track and visualize **metrics**, view model **graphs**, look at **images**, **text** and **audio** data, integrated with TensorFlow and PyTorch.
 - ⇒ **Dashboard by Weights & Biases**: Incredible platform for *tracking your machine learning experiments* (very important) in a few lines of code. See all experiments in **one centralized dashboard** when you're finished.
 - ⇒ **neptune.ai**: Log experiments, model performance, data versions, notebook changes and more.
 - [3] **Data and MODEL TRACKING (what changes have you made to the data? how did they effect the models?):**
 - ⇒ **Artefacts by Weights & Biases**: version your datasets, track your different **ML pipelines**, reproduce your previous datasets.
 - ⇒ **DVC (data version control)**: open-source version control system for machine learning projects (tracks models and data sets).
 - [4] **Computer Services & Hardware:**
 - ⇒ **Cloud compute services**
 - ☛ Google Colab - free GPU powered **JUPYTER** Notebooks
 - ☛ Sagemaker - AWS (**AMAZON** Web Services)
 - ☛ AI Platform (formerly ML engine) - **GOOGLE** Cloud Platform (GCP)
 - ☛ Azure Machine Learning - Microsoft **AZURE**
 - ⇒ **Hardware (building your own)**
 - ☛ Which GPU should you use for machine learning? By **Tim Dettmers**
 - ☛ How to build your own deep learning PC and save \$1,000's by **Jeff Chen**
 - [5] **AutoML (automatically building machine learning models based on your dataset) & hyperparameter tuning**
 - ⇒ **TPot**: Python automated machine learning tool that ■ optimizes machine learning pipelines using genetic w programming.
 - ⇒ **Google Cloud AutoML**: works phenomenal, downside is G (usually) can't download the model, so you need to run co API calls to Google for inference.

⇒ **Microsoft Automated Machine Learning**

⇒ **SWEEPS by Weights & Biases:** trial and track a range of hyperparameter experiments and see which ones work » best.

⇒ **KERAS Tuner:** Hyperparameter tuning library for KERAS models.

[6] **Explainability (why did my model do what it did?)**

⇒ **What-if tool:** Compare different machine learning models, ? Visualize inference results, change **data points** and see how the model reacts.

⇒ **SHAP values:** Use **Game Theory** to explain the outputs of your machine learning models.

[7] **Machine learning LIFECYCLE (end-to-end machine learning projects and user-interface building)**

⇒ **Streamlit:** create beautiful data-driven user-interfaces which you can use to demonstrate your work.

⇒ **MLflow:** track experiments, package code, deploy and store models, all open-source.

⇒ **Kubeflow:** provides a framework for deploying machine learning workflows on Kubernetes (Kubernetes is a framework for building containerized applications, e.g., one container does one thing, such as preprocess data, another does another thing such as model data).

⇒ **Kubeflow Pipelines:** user interface based setup for managing machine learning workflows.

⇒ **Seldon:** bridge between DevOps (maintaining the deployment of a software application) and machine learning. Creates a framework for MLOps (DevOps for machine learning).

⇒ A **guide** to production level deep learning (GitHub repo)



What I learned surveying 200 machine learning tools post by **Chip Huyen** (a great overview of the machine learning tooling landscape, in short, brining research to production is hard).



Libraries (Python flavoured)

[1] **Scikit-Learn:** Extensive machine learning library with features for *preprocessing* data, *modeling* data and *evaluating* models.

[2] **PyTorch:** open-source deep learning library with capabilities for preprocessing data, modelling data and serving models.

⇒ **PyTorch Lightning** (simplified but still just as powerful version of **PyTorch**).

[3] **TensorFlow:** Open-source machine learning framework with capabilities from server to embedded devices.

⇒ **TensorFlow.js:** deep learning in the web.

⇒ **TensorFlow Lite:** on-device inference (mobile & IoT).

⇒ **Keras (high level API)**, now integrated into TensorFlow 2.x+

[4] **ONNX (also in C++):** Open **Neural Network Exchange**, designed to have interoperability between different neural network frameworks. For example, export a **PyTorch** model and run it on the same hardware you'd also like to run a **TensorFlow** model.

1.4 Machine Learning MATHEMATICS (what's running Is. under the hood when you write machine learning code)



Linear algebra: Creating objects and a set of rules to manipulate these objects. E.g. x^2 . x is the object, and 2 is manipulating that object. Machine learning is about finding the right *set of objects* and right *set of rules* to model a dataset.



Book: Rachel Thomas Computational **Linear Algebra**.



Matrix manipulation: In machine learning data (all kinds of it) often gets turned into **rows**, **columns** and **features** (features is the 3rd dimension, which can actually be many dimensions) of numbers. These collections of numbers are often referred to as *matrices* or *tensors*.



Multivariate calculus: Foundation for **optimizing** a function (for example **cost function**) with respect to **multiple parameters** (the patterns a machine learning model learns).



Probability distribution: A collection of probabilities, including, a *sample space*, a series of *possible events*, the probability of an *event* and the (unpredictable) *random event*.



Probability: the study of uncertainty.



Optimization: if machine learning is about finding the most ideal patterns which describe a dataset, how do you w optimize a model to do so?



The Chain Rule: Basis of back propagation, how *neural networks improve* themselves.

❑ Where can I learn all about these:

- ⇒ **The Mathematics for Machine Learning Book:** (Tip) read this book in parts, when you stumble across a new mathematical concept, refer to the chapter on it in this book, read it (even if you don't understand it) and then revisit your problem.
- ⇒ Deep Learning Book
- ⇒ **fast.ai** deep learning from the foundations course
- ⇒ Various other resources (see the resources branch)

1.5 Machine Learning RESOURCES (how to learn, places to visit, etc)

❑ Concepts and process

- ▶ **Elements** of AI
- ▶ **Googles** Machine Learning **Crash Course**
- ▶ **Google's** AI education **page**
- ▶ **Facebook's** field guide to machine learning
- ▶ **Made with ML topics:** A comprehensive collection of the most up-to-date resources for learning about different machine learning topics.
- ▶ **Software 2.0**

❑ Test your skills:

- ☞ **Wokera.ai:** A series of questions by the **deeplearning.ai** team on what kinds of things you should know as a machine learning engineer or data scientist.
- ☞ **Kaggle competitions:** Get data from the real world and test your model building skills while competing with others around the world. A really cool follow on from these projects would be to **deploy your models in a user-facing app** using something like **Streamlit**.

❑ Code

☞ **BEGINNER (if you're completely new, start here):**

- **Python:**
 - 🕒 Learn Python in 1-video on **YouTube** (by **freeCodeCamp**)
 - 🕒 **Zero to Mastery Python** Course (**full-stack Python**), video based course.
 - 🕒 Python like you mean it: Learn Python for STEM applications (data analysis, machine learning, numerical work, etc.)
- **Tooling (barebones):**
 - 🕒 Package management (take care all of your Python packages)
 - ▶ **Anaconda/Miniconda/conda**
 - ▶ **Python virtualenv**
 - 🕒 **Jupyter Notebooks** (write and explore machine learning and data science code)
- **Give me it all in one place:**
 - 🕒 **Zero to Mastery Machine Learning Course** (learn Python, the foundations of machine learning, tools and concepts all in one course). Disclaimer: this is taught by the person (**Daniel Bourke**) who made this roadmap.
 - 🕒 **Kaggle** learning centre
 - 🕒 **DataCamp**
 - 🕒 **Dataquest**

☞ **ADVANCED (after 3-6 months+ of beginner work) or, you've had prior experience as a Python developer/have a mathematics background.**

- **Example project:**
 - 🕒 **End To End Regression** machine learning project by **Aurelien Geron** (author of Hands-on Machine Learning co Book)
 - 🕒 Example Scikit-Learn **Data Exploration And Modelling** by **Daniel Formosso**
 - 🕒 Example **Binary Classification** project by **Daniel Bourke**
- **deeplearning.ai** curriculum (includes Andrew Ng's famous machine learning course and much more)
- **fast.ai curriculum:**
 - 🕒 Overview
 - 🕒 **fast.ai Part 1** (deep learning for coders)
 - 🕒 **fast.ai Part 2** (deep learning from the foundations)

- **CS50's Introduction to Artificial Intelligence with Python:** phenomenal coverage and introduction to many of the most important topics in artificial intelligence.
- **Full-stack deep learning curriculum** (don't let your models die in **Jupyter Notebooks**, get them **live**)
- Get proficient with at least 1 **cloud service** (see below)

☞ **What's missing (general software engineering practices left out of ML curriculums):**

- ❖ Docker
- ❖ Missing parts of your **CS degree**
- ❖ teachyourselfcs.com

□ **Mathematics**

- [1] **Linear algebra:** Rachel Thomas computational linear algebra
- [2] **Matrix** manipulation
- [3] **Calculus:** Multivariate calculus (used in optimization)
- [4] **Neural networks**
- [5] **Probability** and **statistics**

□ **Books**

- [1] **Python:** Automate the boring stuff with Python
- [2] **Data Science:** Data Science Handbook by Jake VanderPlas
- [3] **Data analysis and manipulation:** Python for Data Analysis by Wes McKinney (creator of pandas library)
- [4] **Machine learning practices & code**
 - ➞ **Hands-on Machine Learning with Scikit-Learn, TensorFlow & Keras** by Aurelien Geron (2nd edition)
 - ➞ **Deep Learning for Coders** (with fastai and PyTorch) by Jeremy Howard and Sylvain Gugger (released July 2020)
 - ➞ **Introduction to Machine Learning with Python** by Andreas C. Muller and Sarah Guido
 - ➞ **Mechanics of Machine Learning** by Terrence Parr and Jeremy Howard
 - ➞ **Building Machine Learning Pipelines** by Hannes Hapke & Catherine Nelson (released August 2020)
 - ➞ **Interpretable Machine Learning** Book by Christoph Molnar (explores how to make machine interpretable for structured data)
- [5] **Mathematics**
 - ➞ **Math for Machine Learning** Book by Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong
 - ➞ **The Matrix Calculus You Need For Deep Learning** by Terrence Parr and Jeremy Howard

□ **Cloud Services**

- ☯ **A Cloud Guru**
- ☯ **Google Cloud Training** Resources
- ☯ **AWS Training** Resources
- ☯ **Microsoft Azure Training** Resources (some amazing paths here)

□ **Rules and tidbits**

- 🕸 **Creating a blog:** Advice for creating your own blog (yes, you should have one)
 - ✂ fastpages (blog from Jupyter Notebooks)
 - ✂ GitHub Pages
- ☞ **Andrei Karpathy's** (head of AI at Tesla) recipe for training neural networks
- ☞ **Machine learning 101** (actually 43) rules (e.g. don't use machine learning if you don't have to)... machine learning rules
- ☞ **Practical Advice** for Building Deep Neural Networks (a handful of hands-on tips for deep learning)

□ **Bookmarks**

- 🔗 **arXiv-sanity:** A tool for monitoring the latest **research** from **arXiv**.
- 🔗 **Made with ML: Community driven resource** for projects built with **machine learning** (you should put yours here!).
- 🔗 **Papers with Code:** Most recent machine learning research with code resources.
- 🔗 **sotabench:** State of the **art benchmark tracking** (what is the current state of the art for a series of benchmarks).

□ Datasets

- 👾 **Google Dataset Search:** use the power of Google search to find different datasets.
- 👾 **Kaggle Datasets:** get open machine learning datasets as well as examples of how to work through them.
- 👾 Curated list of free datasets by **Dataquest**
- 👾 **Open Images:** massive repository of *open source images* and labels.
- 👾 **Awesome data labelling:** a curated list of resources for labelling data, including *image annotation*, *audio annotation* and more.
- 👾 **The big bad NLP dataset:** a collection of 545- datasets for *NLP*.

□ Cool resources, how do I actually learn all of this?

- 📺 Learning how to learn on *Coursera*
- 📺 6 Techniques Which Help Me Study Machine Learning *Five Days Per Week* by *Daniel Bourke* (the one who wrote this roadmap)
- 📺 *My Self-Created AI Masters Degree* by *Daniel Bourke* (mostly ML)
- 📺 *How I learned to code* by *Jason Benn* (ML + software engineering + web development)
- 📺 Create your own curriculum