Machine Learning

- Machine Learning Problems
  - 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 (like spreadsheets, database tables, CSV files, or Excel spreadsheet). Can contain different types of data, for example, numerical, categorical, time series.
        - 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.
        - Numerical
          - Any continuous value where the difference between them is meaningful. For example, when selling houses, $107,850 is more than $56,000.
        - Ordinal
          - Data which has order but the distance between values is unknown. For example, questions such as, how would you rate your health? 1, 2, 3 or 4 or how much do you like cars? You can answer 1, 2, 3, or 4 but the distance between each value doesn't necessarily mean an answer of 3 is 5 times as good as an answer of 1.
        - 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).
  - 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, we might have a dataset which contains a single row, 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 in machine learning model can't learn on data that isn't there.
        - Single imputation: Fill with mean, median of column.
        - Multiple imputation: Model other missing values and fill with what your model finds.
        - KNN (k-nearest neighbors): Fill data with a value from another example which is similar.
        - Many more, such as, random imputation, last observation carried forward (for time series), meeting window, most frequent.
      - Feature encoding (turning values into numbers): A machine learning model requires all values to be numerical.
        - 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 given car's color feature would be represented as [1, 0, 0] and a red one would be [0, 1, 0].
        - 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.
        - Embedding: Learn a representation amongst all the different data points. For example, a language model could learn that similar 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. age in years, income in dollars), it can be hard for a model to find patterns (if land between 0.001 and 20,000 as feature values). Many machine learning algorithms don't perform very well. Scaling and standardization help to fix this.
        - Feature scaling (also called normalization): shifts your data so that it's between 0 and 1. This can be done by subtracting the min value and dividing by the max minus the min. For example, 1 and 5 bathrooms would become something like 0.2 and 0.8 respectively (between 0 and 1 but the numerical relationship is still there).
        - Feature standardization: standardizes all values so they have a mean of 0 and a standard deviation of 1 by subtracting the mean and dividing by the standard deviation of that particular feature. Doing this means values do not end up between 0 and 1 (their range is unscapped). 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 22:14:34) into hour_of_day, day_of_week, day_of_month, is_holiday, etc.
        - Discretization (turning larger groups into smaller groups).
          - For numerical variables, let's say age, you may want to move it into buckets, such as, over_30 or under_30, Or, 0-20, 21-40, 41-60, 61-80, 81-100 (this 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 is_to_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? For car sales, is_luxury_and_manual_transmission for cars that have less than 100,000 RRP, 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) looks at the variance of different features and tries to 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 good set of features but 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 hyperparameters are tuned on this.
      - Test set (usually 10-15%): Models final performance is evaluated on this. Ideally you don't touch this until results on the test set give a good indication of how the model should perform in the real world. Do not use this dataset to tune the model.
  - Train model on data (3 steps: choose an algorithm, overfit the model, reduce overfitting with regularization)
    - Choosing an algorithm
      - Supervised algorithms
        - 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 Neighbours
          - Find the 'k' examples which are most similar to each other. Then, given a new example, which is the new sample most closely aligned with?
        - Support Vector Machines (SVMs)
          - Can be used for classification or regression. Try to find a line (called a hyperplane in 2 dimensional space, or referred to as Hyperplane). 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 (very flexible algorithms). Decision trees look at different data 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. Ensembling (combining many models) 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. Builds 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 way of ensembling, this time ensembling 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) and non-linear functions (called activation functions) functions. 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 modelling).
            - Transformer networks (can be used for vision and text, starting to replace RNNs).
      - Unsupervised algorithms
        - Clustering
          - K-Means clustering: choose 'k' number of clusters, each of which has a centroid (average). Random points are chosen 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 more dimensions to lower dimensions whilst attempting to preserve the variance.
          - Autoencoders: 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 high-dimensionality data in a 2D or 3D space.
        - Anomaly detection
          - Autoencoder: Use an autoencoder 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 of data (for example, transactions which are generally usually in abundance). If anything falls 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.
    - Overfitting
      - Regularization: a collection of techniques to prevent/reduce overfitting.
        - L1: Sparsal and L2 (ridge) regularization. L1 regularization sets unwanted feature coefficients to 0 (performs feature selection on which features are most essential and which aren't, useful for model explainability). L2 contrains 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 trying to build a computer vision model, you can adjust the height of your images. This makes your model have to learn similar patterns across different styles of the same image (harder). Relax since this can be compute intensive, it's a good idea to do this in memory. See a PyTorch library, ImageDataGenerator in Keras or transforms in torchvision.
        - 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 for dropout in some networks.
      - Hyperparameter Tuning (run a bunch of experiments with different model settings and see which works best)
        - Skry: useful paper! A disciplined approach to neural network hyperparameters by Leslie Smith. Covers learning rate, batch size, momentum, weight decay and more.
        - Setting a learning rate (often the most important hyperparameter): generally, high learning rate + algorithm adapts slower 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. 1e-5) and finishing with a very high learning rate (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. See p226 Hands-on Machine Learning Book Edition 2.
            - Learning rate scheduling (see Adam optimizer) involves lowering 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.
        - 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 LeCun's Twee), 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_name2 hyperparameter tuning'.
  - Analysis/Evaluation
    - Evaluation metrics
      - Classification
        - Accuracy
        - Precision
        - Recall
        - F1
        - Confusion matrix
        - Mean average precision (object detection)
      - Regression
        - MSE (mean squared error)
        - MAE (mean absolute error)
        - R^2 (r-squared)
      - 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".
    - 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
      - MLflow: where software engineering meets machine learning, essentially all the technology required around a machine learning model to have it working in production.
  - Serve model (deploying a model)
  - Retrain model
    - See how the model performs after serving (or prior to serving) on new, previously unseen data (data that wasn't 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 time-wise style) or 'drift', as in when data sources change or upgrade (new hardware, etc). This is when you'll want to retrain it.
- Machine Learning Resources (how to learn, places to visit, etc)
  - See full interactive version at: https://dbourke.link/mlmap
  - Machine Learning Process (steps in a machine learning project)
  - Books
    - Machine learning practice & 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 (explains how to make machine interpretable for structured data)
    - 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
  - Bookmarks
    - Rules and tidbits
      - Machine learning 101 (actually 43 rules (e.g. don't use machine learning if you don't have lot... machine learning rules
      - Practical Advice for Building Deep Neural Networks (a handful of hands-on tips for deep learning)
      - 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).
      - Google Dataset Search: use the power of Google search to find different datasets.
      - Kaggle Datasets: an open machine learning datasets as well as examples of how to work through them.
      - Curated list of free datasets by Datasquat
      - 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 5x5= datasets for NLP.
  - Datasets
    - Cool resources, how do I actually learn all of this?
      - Learning how to learn on Coursera
      - 6 Techniques Which Help Me Study Machine Learning Fine (Days Per Week by Daniel Bourke (the one who wrote this roadmap)
      - Create your own curriculum
        - My Self-Created AI Masters Degree by Daniel Bourke (mostly ML)
        - How I learned to code by Jason Benn (ML + software engineering + web development)
    - Creating a blog
      - Fastpages (blog from Jupyter Notebooks)
      - Github Pages
    - Advice for creating your own blog (yes, you should have one)
    - Andrei Karpathy's (head of AI at Tesla) recipe for training neural networks
  - Toolbox
    - Libraries (Python flavoured)
      - Scikit-Learn: extensive machine learning library with features for preprocessing data, modelling data and evaluating models.
      - PyTorch: open-source deep learning library with capabilities for preprocessing data, modelling data and serving models.
      - TensorFlow: open-source machine learning framework with capabilities from server to embedded devices.
      - ONNX (stands for Open Neural Network Exchange, pronounced "onnicks"): a standard for deep learning 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.
      - TensorFlow Hub
      - PyTorch Hub
      - HuggingFace Transformers (for natural language processing)
      - Detectron2 (for computer vision)
    - 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.
    - Cloud compute services
      - Google Cloud: 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 by Jeff Chen
    - Pretrained models (for transfer learning)
    - 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)
    - AutoML (automatically building machine learning models based on your dataset) & hyperparameter tuning
      - 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.
    - 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.
    - Machine learning lifecycle (end-to-end machine learning projects and user-facture building)
      - Streamlit: create beautiful data-driven user-interfaces which you can run 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)
  - Machine Learning Mathematics (what's running under the hood when you write machine learning code)
    - What I learned surveying 200 machine learning tools pool by Chip Huyen (a great overview of the machine learning tooling landscape, in short, brining research to production is hard)
    - Linear algebra (creating objects and a set of rules to manipulate these objects. 2, 3, or 5, 2 is an object, and 1/2 is manipulating that object. Machine learning is about finding the right set of objects and right set of rules to model a dataset.
      - 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 (the 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 optimize a model to do so?
      - The Chain Rule: basis of backpropagation, how neural networks improve themselves.
    - Rachel Thomas Computational Linear Algebra
    - 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.
    - Where can I learn about all of these?
      - Deep Learning Book
      - fast.ai deep learning from the foundations course
      - Various other resources (see the resources branch)

Made by Daniel Bourke (any questions, feel free to ask).
Copy/use whatever you need.
Daniel Bourke

Inspiration drawn from (you should also check this out):
GitHub - dformoso/machine-learning-mindmap: A mindmap summarizing Machine Learning concepts, from Data Analysis to Deep Learning.

- Categories Types of learning
  - Supervised
    - 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). Works for numbers too, for example, you could use data about houses to try and predict information about the houses to try and predict the selling price.
  - Unsupervised
    - You have data but no labels, the model tries to find patterns in data without something to reference on. For example, you have 10,000 photos of cats and dogs but they are similar but 3 of them are completely outlandish, which (?) this kind of problem is called anomaly detection. Other common problems include Clustering and dimensionality reduction (PCA).
  - Reinforcement Learning
    - An agent (algorithm) performs actions in an environment and is rewarded or penalized based on the whether the actions are moving towards the goal. For example, you could be a (TeamPlayer?) 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's rewarded. Over time, the agent learns (based on the actions it gets rewarded for) to called a policy.
  - Transfer learning
    - Take knowledge from one model and use it on your own. 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.
- Example problems
  - Binary classification
    - Is this email spam or not spam?
    - Is this a photo of a cat or a dog?
  - Multi-class classification
    - Is the traffic light green, yellow or red?
    - What breed of dog is in this photo?
  - Multi-label classification
    - What time 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 eachother. Useful for seeing where a model is getting 'confused'. E.g. how often it predicts the right and wrong classes.
  - Accuracy
    - Out of 100 examples, how many will I get right?
    - Note: correlates when number of samples differ. E.g. 99999 examples of class A and 1 example of class B is a model which predicts all class A is 99.999% accurate. In this case, use precision and recall instead.
  - Precision
    - Accuracy of the positive predictions. Of the positive predictions, what percentage 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.
  - F1 score (combination of precision and recall)
  - Precision/Recall tradeoff
    - Increasing precision reduces recall and vice versus. 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) is more important. 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 99% + recall", ask, "at what precision?".
  - Precision/Recall curve
    - Plot the precision versus recall at different classification thresholds. Allows you to choose a precision value of 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 (you can use this value to compare different classifiers).
- Classification
- Regression
  - Evaluation
    - 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.
  - Example problems
    - 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.
  - Sequence to sequence (seq2seq)
  - Example problems
    - Group the customers of the business by their purchases. Unlike other kinds of problems (such as supervised problems), Strict groups are found, labels can be added.
  - Clustering
  - Example problems
    - Reduce the number of inputs into a model by mapping the original inputs to new inputs. For example, instead of you have 100 inputs (which may be too many for your model), how can you reduce this down to the 10 most important inputs? A 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 representation of a given data input for a machine learning task. For example, have an algorithm read the entirety of Wikipedia to form a representation of how words should appear in the context of others (build a language model). How do you choose (or learn) the best representation of a data a machine learning model can learn on?
  - Dimensionality Reduction
- Elements of AI
- Google's 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.
- Weka: 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).
- Software 2.0
- Beginner (if you're completely new, start here).
  - Tooling (Sandbrewn)
    - Anaconda/Miniconda/conda
    - Python virtualenv
    - Package management (take care all of your Python packages)
    - 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, machine learning, data science, deployment and concepts all in one course). Disclaimer: this is taught by the person (Daniel Bourke) who made this roadmap.
    - Kaggle learning centre
    - DataCamp
    - Datasquat
  - Advanced (after 3-6 months of beginner work) so you've had prior experience as a Python developer/have a mathematics background.
    - Linear algebra
    - Calculus
    - Matrix manipulation
    - Neural networks
    - Probability and statistics
  - Python
    - Automate the boring stuff with Python
    - Data analysis and manipulation
    - Data science
      - Data Science Handbook by Jake VanderPlas
  - Machine Learning Resources (how to learn, places to visit, etc)
    - 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.)
    - deeplearning.ai curriculum (includes Andrew Ng's famous machine learning course and much more)
    - Overview
    - fast.ai Part 1 (deep learning for coders)
    - fast.ai curriculum
    - 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)
    - Docker
    - Missing parts of your CS degree
      - teachyourselfcs.com
    - Rachel Thomas computational linear algebra
    - Multivariate calculus (used in optimization)
    - Matrix manipulation
    - Neural networks
    - Probability and statistics
  - Python
    - Data analysis and manipulation
    - Data science
      - Data Science Handbook by Jake VanderPlas
  - Machine learning practice & 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 (explains how to make machine interpretable for structured data)
  - 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
  - A Cloud Guru
  - Google Cloud Training Resources
  - AWS Training Resources
  - Rules and tidbits
    - Machine learning 101 (actually 43 rules (e.g. don't use machine learning if you don't have lot... machine learning rules
    - Practical Advice for Building Deep Neural Networks (a handful of hands-on tips for deep learning)
    - 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).
    - Google Dataset Search: use the power of Google search to find different datasets.
    - Kaggle Datasets: an open machine learning datasets as well as examples of how to work through them.
    - Curated list of free datasets by Datasquat
    - 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 5x5= datasets for NLP.
  - Learning how to learn on Coursera
  - 6 Techniques Which Help Me Study Machine Learning Fine (Days Per Week by Daniel Bourke (the one who wrote this roadmap)
  - Create your own curriculum
    - My Self-Created AI Masters Degree by Daniel Bourke (mostly ML)
    - How I learned to code by Jason Benn (ML + software engineering + web development)
  - Creating a blog
    - Fastpages (blog from Jupyter Notebooks)
    - Github Pages
  - Advice for creating your own blog (yes, you should have one)
  - Andrei Karpathy's (head of AI at Tesla) recipe for training neural networks
  - Machine Learning Tools Tools (you can use to get the job done)
  - Machine Learning Tools Tools (you can use to get the job done)
  - Machine Learning Tools Tools (you can use to get the job done)