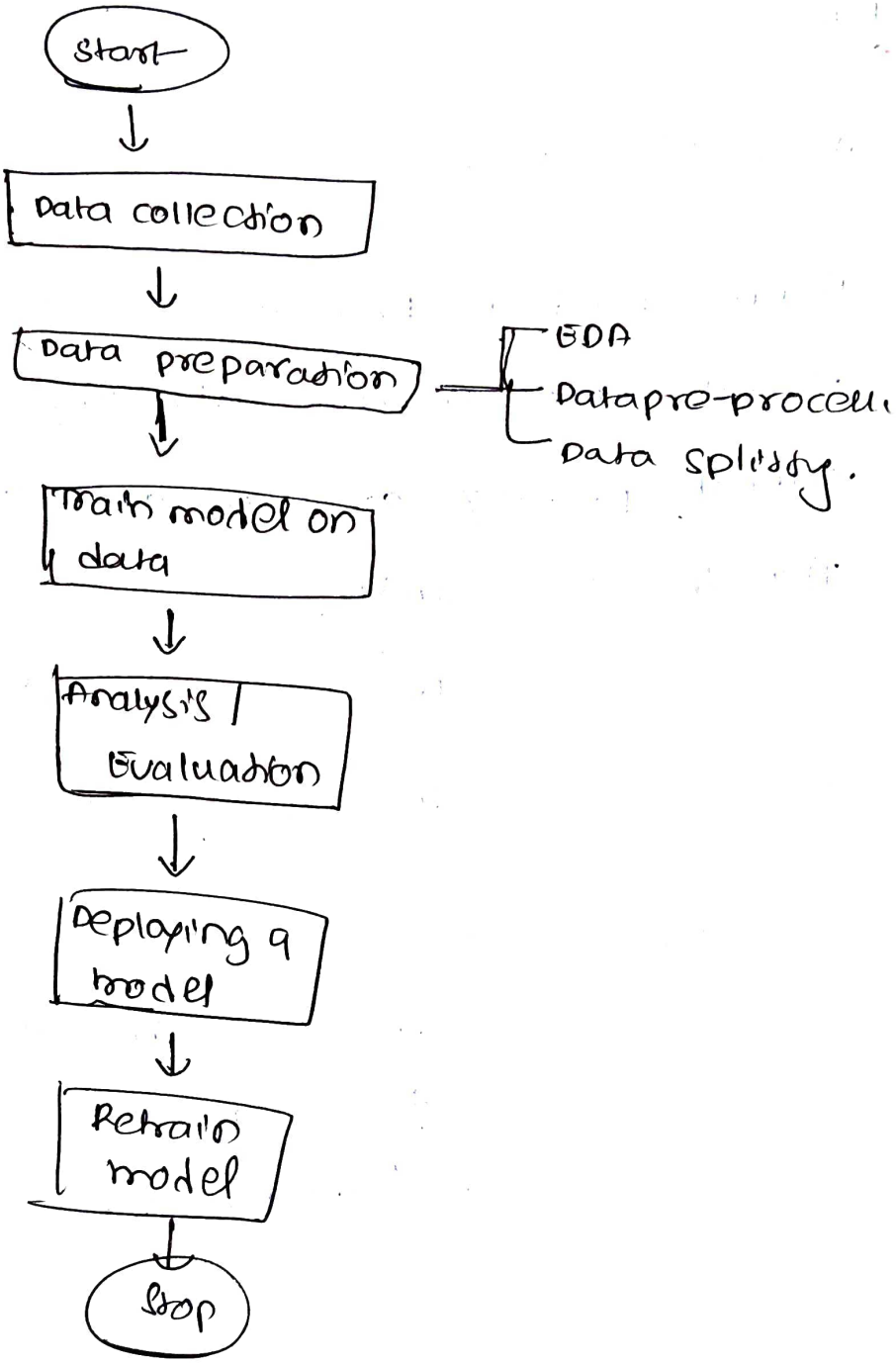
Table Of Contents

4. Exploring Step 3 - Train Model on Dataset
    - 4.1 Types of Learning
    - 4.2 Under Fitting and OverFitting
    - 4.3 Regularization techniques
    - 4.4 Hyperparameter Tuning
5. Exploring Step 4 - Evaluation of a Model
    - 5.1 Evaluation Metrics
    - 5.2 Confusion Matrix
    - 5.3 Recall/Sensitivity
    - 5.4 Precision
    - 5.5 Specificity
    - 5.6 F1 Score
    - 5.7 AUC and ROC Curve
    - 5.8 Analysis of a Model

ML Road map (Steps in machine learning project)



(*) understanding the process :-

Step (1)

Data collection

a) Questions to ask ?

1) what kind of problem are we trying to solve?
2) what data sources already exist?
3) what privacy concerns are there?
4) is the data public?
5) where should we store the data?

b) understanding the types of data

1) Structured data ?

Data which appear to be tabulated format can contain diff types of data.

like
i) nominal / categorical.
ii) numerical
iii) ordinal
iv) time series

2) unstructured data?

Data with no rigid structure.

(Images, video, natural language text, speech)

Step (2)

Data preparation

It has 3 main process.

a) Exploratory data analysis (EDA), understanding data.

b) Data preprocessing, preparing your data to be modelling

c) Data splitting.

---

a) EDA, understanding the data

1) what are feature variables (input) and the target variable (output)?

Eg: For predicting heart diseases, feature variable may be a person's age, weight, heart rate etc. And target variable is whether or not they have heart disease

2) what kind of data do you have?
Structured, unstructured, categorical / numerical.

3) Are there missing values? should you remove them / fill them with feature imputation?

4) what are outliers?
How many of them are there?
Are they out by much (3+SD)?
Why are they there?

5) Are there questions you could ask a domain expert about the data?

b) data preprocessing, preparing your data to be modelled

① Feature imputation:-

(Filling missing values)

→ A ML model can't learn on data that isn't there so,

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 neighbour) :- 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.

∴ there are ways to handle missing values.

② Feature encoding:-

(Turning values into numbers)

→ A ML model requires all values to be numeric

i) OneHotEncoding:-

Turn all unique values into lists of 0's & 1's where the target value is 1 & the rest are 0's.

Eg: car colors green, red, blue,

a green car's color feature would be [1,0,0]

& a red one would be [0,1,0]

& a blue one [0,0,1].

ii) LabelEncoder :-

Turn labels into distinct numerical values

eg:
If your target variables are diff animals as dog, cat, bird then 0, 1, 2 respectively

iii) Embedding encoding :-

Learn a representation amongst all the diff data points.

eg:
A language model is a representation of how diff words relate to each other. embeddings are also becoming more widely available for structured data.

---

(3) Feature normalization (scaling) or standardization :-

when your numerical variables are on diff scales. (like size of land b/w 1000 - 20000 sq.ft).
In such case some ML models don't perform well. Scaling / standardization help to fix this.

i) Feature scaling (normalization) :-

shifts your values so they always appear b/w 0 & 1.

* this is done by subtracting min value & divide with range.
(value b/w 0 & 1 but numerical relationship is still there)

ii) Feature standardization :-

Standardized all values so they have a mean of 0 & variance of 1.

* this is done by subtracting mean & divide with SD.
* values doesn't end up b/w 0-1 (if variance).
* Standardization is more robust to outliers than scaling.

④ Feature Engineering :-

→ Transform data into potentially more meaningful representations by adding in domain knowledge.

i) Decompose

Like, turn a date (such as 2020-06-18 to 20:16: into hour_of_day, day_of_week, day_of_month, is_holi etc..

ii) Discretization

(Turning larger groups to smaller groups)

- For numerical variables,
  Like age you may want to move it into buckets such as over_50 (under_50, (21-30, 31-40 etc.) this process is known as (BINNING).
- For categorical variables,
  such as car color, this may mean combining colors such as (light_green), (dark_green), (wine green) into a single green

iii) Crossing & interaction features

(combining two/more features)

→ depends on situation. Sometime adding / subtracting two features could help.

iv) Indicator features :-

like using is_paid_traffic for paid

[using other parts of data to indicate something potentially significa

→ create a x_missing feature for wherever a column x contains a missing value.

⑤ Feature selection :-

→ selecting the most valuable features of your dataset to model.

→ Potentially reducing overfitting & training time (less overall data & less redundant data to train on) & improving accuracy.

i) Dimensionality reduction :-

A common dimensionality reduction method, PCA takes a larger no. of dimension (features) & uses linear algebra to reduce them to less dimensions. For example, say you have 10 numerical features you could run PCA to reduce it to 3.

ii) Feature importance :-

Fit a model to a set of data, then inspect which features were most important to the results, remove the least important ones. (can use "TPOT" for this)

⑥ Dealing with imbalances :-

Does your data have 10,000 examples of one class, but only 100 examples of another?

→ collect more data (if possible)

→ use scikit-learn-contrib imbalanced-learn Package.

→ use SMOTE (synthetic minority over-sampling technique) which creates synthetic samples of your minor class to try & level the playing field.

→ So this is all about Data Preprocessing.

c) Data splitting

1) training set (usually 70-80% of data):
model learns on this.

2) validation set (typically 10-15% of data):
model hyperparameters are tuned on this.

3) Test set (usually 10-15%): (Dev set)
models final performance is evaluated on this
* do not use this dataset to tune the model

Q. why shouldn't we tune on test set?

A.
To do the regularization we have to take 100 diff models & try on the test data. And so last we finds (suppose) the best hyperparam that produces a model with low generalization error, say just 5%.

And you launch it to production & found it produces 15% error.

* now, here is what happened as you trying every hyperparameter on the same test data eventually your model learned the test data & fitted to it thus reducing generalization error but not improving model.
this is called "overfitting"

→ so to avoid we take validation set / cross-validation

Step ③

Train model on dataset

It has 3 main process.

a) choose an algorithm
b) overfit the model
c) Reduce overfitting with Regularization.

---

a) Algorithm can be choosed by understanding supervised, unsupervised (explained previous)

b) Types of learning :-

i) Batch Learning :-

All of your data exists on a big static warehouse & 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 & isn't done often. Runs in production without learning (can be retrained).

ii) Online Learning :-

Your data is constantly being updated & you constantly train new models on it. Each learning step is usually fast & cheap.
Runs in production & learns continuously.

iii) Transfer learning:-

Take the knowledge one model has learned & use it with your own. Gives you the ability to leverage SOTA (state of the art) models for your own problems.
Helpful if you dont have much data / vast compute resources.

iv) Active learning:-

Also referred to as "human in the loop" learning. A human expert interacts with a model & provides updates to labels for samples where the model is most uncertain about.

v) Ensembling:-

Not really a form of learning, more combining algorithms which have already learned in some way to get better results.

Underfitting:-

Happens when your model doesn't perform as well as you'd like on your data. Try training for a longer & more advance 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".

(eg:- 99% accuracy on training set, 67% accuracy on test set) → overfitting the training set

For through various regularization techniques.

Regularization :- [techniques to prevent/reduce overfitting.]

i) L1 (Lasso) & L2 (ridge) regularization :-

→ L1 regularization sets unneeded feature coefficients to 0.

→ L2 contains a models' features (won't set them to 0)

ii) Dropout :-

Randomly remove parts of your model. So the rest of it has to become better.

iii) Early Stopping :-

Stop your model from training before the validation loss starts to increase too much / more generally, any other metric has stopped improving. Early stopping usually implemented in the form of a model callback.

iv) 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 & adjust the height of your images.

→ this makes your model have to learn similar patterns across diff styles of the same image (harder).

→ use functions like ImageDataGenerator in keras or transforms in torchvision.

---

v) Batch Normalization :-

Standardize inputs as well as adding two parameters (beta, how much to offset the parameter for each layer & epsilon to avoid division by zero) before they go into next layer. this often results in faster training speed since the optimizer has less parameters to update.

maybe a replacement for dropout in some networks.

Hyperparameter Tuning

→ Run a bunch of experiments with diff model settings & see which works best.

a) Setting a learning rate:

(often the most important hyperparameter)

Generally

High learning rate = algorithm rapidly adapts to new data

Low learning rate = algorithm adapts slower to new data

(est for transfer learning)

i) Finding the optimal learning rate

Train the model for a few hundred iterations starting with a very low learning rate (10^{-6}) & slowly increase it to very large value (e^{+1}) then plot the loss vs learning rate (using a log-scale for learning rate), you should see a U-shaped curve, the optimal learning rate is about 1-2 notches to the left of the bottom of U-curve. (P326 - Hands on ML)

ii) Learning rate scheduling involves decreasing the learning rate slowly as model learns more (gets closer to convergence). [like Adam optimizer]

iii) Cyclic learning rates

Dynamically change the LR up & down b/w some thresholds & potentially speed-up training.

Other hyperparameter you can tune

1) No. of layers (Deep learning networks)
2) Batch size [How many eg of data your model sees at once.]
3) No. of trees (Decision tree algos)
4) No. of iterations [How many times model goes through the data]
... Many more ...
Depends on algo we use. (can use early-stopping)

Step ④

Analysis / Evaluation:- to estimate how well a model will generalize to out-of-sample data.

a) Evaluation metrics:-

i) classification
- ROC & AUC curve
- Accuracy
- Precision
- Recall
- F1 score
- confusion matrix.

ii) Regression
- MSE (mean squared error)
- MAE (mean absolute error)
- R2 (r-squared)

iii) Task-based metric (create one based on your specific problem).

1) **Classification accuracy:-**

Percentage of correct predictions. $\left(\frac{Y\_pred}{Y\_test}\right)$

Null accuracy:-

Accuracy that could be achieved by always predicting the most frequent class.
- It's a good way to know the min. we should achieve with our models.
- It gives a baseline to compare our model's accuracy.

∴ classification accuracy is easy, but it doesn't tell you the underlying dist of response class.
- And it doesn't tell you what type of error your classifier is making.

2) confusion matrix

FP - False Positive
FN - False negative.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

TP: which are actually correct

TN: which are actually wrong

FP: they are actually wrong, but predicted as true.

FN: they are actually true, but predicted as False

they actually don't have corona. But we predicte they have

they actually have, but we say they don't

→ classification accuracy ⟨-⟩ $\frac{TP+TN}{TP+TN+FP+FN}$ talks about correct.

How often is the classifier correct?

→ misclassification rate ⟨-⟩ $\frac{FP+FN}{TP+TN+FP+FN}$

Recall / sensitivity [True positive rate]

→ How "sensitive" is the classifier to detecting positive instances?

→ when the actual positive is tested, how often it is correct?

Predicted Pos.

TP

$$\text{Recall} = \frac{TP}{TP+FN}$$

→ It depends on FN, which were actually true, but predicted false.

out of total positive ones, How many positive I am able to (Recall)

Actual Pos - TP+FN

Precision & Recall linked with TP

Precision & — Really Pos / Total Pos: actually

TP / TP+FP — Really Pos

TP / TP+FN — Predicted Pos

→ when a positive value is predicted, How often is the prediction correct?

→ How "precise" is the classifier when Predicting positive instance.

out of Predicted Pos, How many really pos.

$$\text{Precision} = \frac{TP}{TP+FP}$$

→ It depends on FP, which were actually negative, but predicted true.

Q.) which metrics to focus on?

Eg ①

1 = COVID 19 (+)

0 = − (Healthy)

Actual \ Pred	0	1
0	TN	FP → Healthy predicted as sick..
1	FN → Sick predicted as healthy :[	TP

→ Now analyze the cost

→ we know a sick predicted as healthy could spread a lot more cases.

So, cost of FN > cost of FP

Hence It depends on FN, we go for "Recall".

Eg ②

1 = Spam

0 = Not Spam

Actual \ Pred	0	1
0	TN	FP → Not Spam predicted as Spam :(
1	FN → Spam got into mail box.	TP

→ we know if a not Spam (imp. mail) goes to Spam, that could cause lot of damage.

So, cost of FP > cost of FN

Hence, It depends on FP, we go for "precision"

Specificity :-

$$\frac{TN}{TN+PP}$$

False positive rate :-

$$\frac{PP}{TN+PP}$$

FB-Score :-

when you want to consider both PP & FN like we need both sensitivity (recall) & precision, in such cases we choose F-Beta.

General Formula

$$F_B = (1+B^2) \cdot \frac{Precision \times recall}{(B^2 \cdot Precision) + recall}$$

$$F_B = \frac{(1+B^2) \times TP}{(1+B^2) \times TP + B^2 \cdot FN + FP}$$

In beta, there is posibility for 3 ranges.

1) when FP & FN are equally important, like you can't compromise even a bit both are crucial, then we set beta = 1 which is F1-score.

$$F1\text{-}Score = 2 \times \frac{Precision \cdot Recall}{Precision + Recall}$$

2) when both are worse, but FP is drastic than FN. then we go for beta = 0.5 (usually 0-1)

3) when both are worse, but FN is drastic than FP. then we go for beta = 2 (usually 1-10).

---

note

we can also adjust classification threshold to modify performance of classifier.

Roc curve

can see how sensitivity & specificity are affected by various thresholds, without actually changing the threshold.

AUC curve (Area under curve)

If you randomly choose one + & one -, AUC represents the likelihood that your classifier will assign a higher predicted probability to the positive observation.

Analysis of model

1) Feature importance

Which feature contributed most to model
Should some be removed?

2) Training / Inference time

How long does a model take to train?
Is this feasible?

How long does inference take?
Is it suitable for production?

3) Using "what-if" tool

what-if changed thing in data?
How does this effect the outcome?

4) Bias / variance trade-off

→ High bias results in underfitting &
a lack of generalization to new samples

→ High variance results in overfitting du
to the model finding patterns in the
data which is actually random
noise.

Step (5)

Deploying a model

Put the model into production & see how it goes. Evaluation metrics in notebook are great but until it's production, you won't know how it performs for real.

Step (6)

Retrain model

→ See how the model peforms after serving based on various evaluation metrics & revisit the above steps as req.

→ You will find your models prediction start to cages or (drift), as in when data sources change or upgrade. this is when you will want to retrain it.

(0,0)

Stop / NO END

So, basically the flowchat is ---: