**Real World Data Science Challenge Steps:**

* Formulize the problem
* What choice of evaluation
* What data to be used
* Build models
* Evaluate the model

***Learning Models***

*There is no free lunch. Each model expose its assumption on data and reduces generalizability. Mosst widely used are XGBoost and NN.*

**Linear models:** separates with linear lines

* Pro: Good with sparse high dimensional data
* Con: Too simple and might introduce bias
* Ex: SVM, linear regression, logistic regression
* Libr: Sikitlearn, Vowpal
* Consideration: requires feature scaling, outlier removing

**Tree based models:** divide and conquer approach

* Pro: flexibility in assembling, very good starting points, no need for sacling
* Con: don’t do well with linear data
* Ex: Decision trees, random foreset, gradient boosting trees
* Libr: Sikitlearn, XGBoost, LightGBM

**Nearest neighbors:** looking at neighbors for insight

* Pro: can be used for feature generation, fast
* Con: don’t do well with linear data
* Ex: K-NN, K-means clustering
* Libr: Sikitlearn
* Consideration: requires feature scaling, outlier removing

**Neural networks:** looking at neighbors for insight

* Pro: smooth boundaries, black box, good for image, sound, text, sequence
* Con: don’t do well with linear data
* Ex: convolutional neural network,
* Libr: Tensorflow, keras, pytorch, lasgna
* Consideration: requires feature scaling, outlier removing

**Random Forests:** an ensembler for decision trees

Randomized based on the number of columns and rows included

* Pro: bias stays at the level of DT, while variance decreases, quick and dirty solution, good for feature selection
* Con: features are not interpretable, not good for smaller data sets. In regression, range is limited to available data.
* Libr: Sikitlearn

***Feature Processing***

Depending on the type of feature, there are some well-known techniques. Choice of model is also influential.

**Numerical data:**

[**Processing**](http://scikit-learn.org/stable/modules/preprocessing.html):

* [**Scaling**](http://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html): sklearn.preprocessing.StandardScaler
* **Clipping**: numpy.clip (for outliers)
* **Rank transformation**: scipy.stat.rankdata (for outliers)
* **Transformation**: log, sqrt (shrinking outliers)

**Engineering**:

* **Interaction**: \*-/+ between features (dig data)
* **Decimal points**: 4.99🡪0.99
* **Group statistics:** std, mean, max, min of the group this observation is in (use groupby, agg).

**Categorical data:**

[**Processing**](http://scikit-learn.org/stable/modules/preprocessing.html):

* [**Label**](http://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html) **Encoding:** generate labels from 0 to n-1 level, good for ordinals and linear models. Sklearn.preprocessing.labelencoder
* **Factorization:** for non-ordinals, pandas.factorize
* **Frequency encoding:** giving the ratio of the number of that class as the value for it, when frequency might be correlated with output. Use rank when some frequencies are too close.
* **OneHotCoder:** generates scaled dummies, good for linear classifiers, and non linear with target, sklearn.preprocessing.onehotencoder

**Engineering**:

* **Interactions**: .str+.str
* **Mean encoding**: each class has a value equal to mean of target in that class multiplied by its encoded label.

**Datetime data:**

**Engineering**:

* **Time moments and periods**: year, month, …
* **Time past since or until**:last holiday, last time bought
* **Time delta between two datetime columns**,

**Text data:**

[**Processing**](http://scikit-learn.org/stable/modules/preprocessing.html):

* [**lowercasing**](http://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html): .lower()
* **Lemmatization/stemming:** getting the roots of words
* **Stopword cleaning:** taking repetitive words out

**Engineering**:

* **Bag of words:** one column for each word and see its repetition in each row, sklearn.fature\_Extraction.text.Countvectorizer
* **TFIDF,** normalizing words bag of words, both row-wise and column wise (how important is feature in that sentence, and how repetitive is it in all data)

Sklearn.feature\_extraction.TfdifVectorizer

* **N-gram:** bag of words for N neighbor words, CountVectorizer has “n-range”, use it for vectorizing.
* **Word2vec:** transfers words to a complicated space. Meaning of vectors are not known, but space is much smaller than bag of words. Similar vectors have similar meaning.

**Image data:**

**Engineering**:

* **CNN:** similar to word2vec but for images, you can either train from scratch or use another network withsimilar object and **finetune** the model.

**Coordinate data:**

**Engineering**:

* **Distance to an important point, old building,**

**Handling missing values:**

Look into histogram, any unusual bump might be missing values encoded.

**Imputations:**

* A value far out of the min and max range, (good for tree based)
* Mean, median (good for non-trees)
* A categorical value (is\_missing)

**Feature Selection**

* **Unsupervised learning:** Use PCA,NMF,SVD to choose feature space with more info,
* **Feature importance:** use Random Forest, XGBoost, to get the feature importance

***EDA and Data Cleaning***

Helps understanding the data and getting it ready for deploying to classifiers.

**EDA Tips:**

**Higher level understanding**:

* df.dtypes(), df.info(), df.describe(), df.isnull(), d.head(), df.tail(), select\_dtypes(include=[int])

**Visualization**

* single variable: plt.hist(), plt.plot(df, ‘.’) (good if standard scaled, to see how data changes through all rows)
* multiple variables: plt.scatter(f1,f2,c=target), plt.matshow(df.corr()), df.mean().sort\_values().plot(style=’.’)

**Cleaning Tips:**

* dropping rows with similar value in all rows: df=[~df.nunique(axis=1)==1]
* df.T.drop\_duplicates()
* for categorical duplicates with different names: for f in categorical\_feat: dff]=df[f].factorize() then the above comment
* select\_dtypes(include=[int])

***Validation and overfitting***

Does model generalize well?

**Validation:**

**Types:**

* Holdout: dividing into train test
* K-fold: repeated hold-out then average
* Leave one out: only when model is very fast, you can have k=len(data)

Make validation stratified if the classes are skewed and there are many features in one vs other

**Data split strategies:**

Validation should mimic the train test split (this is important for kaggle). But for real-world, validation should mimic real world. Types of validation:

* Random split
* Time-wise split
* By id

**Data leaks:**

(related to kagge) Unexpected info in data that allows to estimate the target without building a model. Types of data leaks:

* Time series: future picking,
* Meta data: e.g. time and place pictures taken

**Metrics:**

**Metric vs Loss:**

Metric is how we assess the quality of technique, loss is what model tries to minimize. Sometimes they overlap (like regression MSE).

**Validation (evaluation) metrics:**

We need to estimate how we work on real-world situations. Different metrics results in different hyperplanes.

1. **Regression Metrics**:

* Mean square error(**MSE**). Mean value minimizes it, default for most regression,
* Root mean square error (sqrt(MSE))
* **R-squared**, how much better than a baseline (y=mean) MSE prediction we are?
* Mean absolute error (**MAE**), where errors are linearly penalized. Median minimizes it.
* Mean square percentile error(**MSPE**), relative error minimization, depending on the size of target (error/target is minimized), weighted mean of target value minimizes it,
* Mean absolute percentage error(**MAPE**), weighted median of target value minimizes it.
* Root mean square logarimic error, weighted mean in logspace minimizes it.

1. **Classification metrics**:

* **Accuracy**: fraction of correctly predicted, non-immune to skewed classes, non-caring about level of confidence in the prediction
* **Logloss**: requires soft prediction (probability of target). Penalize when confident and wrong. Class ratio (class A/total) minimizes it.
* **Are under curve**: tries all possible threshold and find the accuracy and aggregate on it. Its base is 50%.
* **Cohen’s Kappa**: good for skewed classes, 1-(1-accuracy)/(1-baseline). You need to combine weighted error matrix and confusion matrix.

**Metric optimization:**

* Models optimize loss, not metrics. MSE and Logloss can be both metric and loss. But the rest needs optimization.

**Early Stopping, golden method for optimization:**

* We keep training the model and calculate the cross validation metrics, until the metric doesn’t show real improvement.

***Hyperparameter Tuning***

Each model has parameters to fit. Increasing or decreasing them might result in overfit(in green) or underfit(in red) the model.

**XGBoost, LightGBM**

* Maxdepth (hoe many level), subsample (bagging fraction), colsample\_by\_level, colsample\_by\_tree, eta (learning rate ), num\_round (how many iteration)
* Min\_child\_weight (min data in leaf) most important parameter

**RandomForest**

Random forest trees are independent as opposed to XGBoost.

* N\_estimator : number of trees, makes model better and saturate in a place.
* Max\_depth, how much deep you can go,
* Max\_featue: number of features used in each tree
* Min\_sample\_leaf
* Criterion: gene works good
* N\_jobs: number of cores on your machine.

**Neural Networks**

* Number of neurons per layer, number of layers, batch size,
* Optimizer: either choose gradient descent + momentum, or adam
* Learning rate,
* Neural net regularizes:
  + L2/L1 for weights
  + Dropout, drop connections (do this close to the last layer)
  + Statistic drop connect

**SVM**

* Regularization, regularization type.

***Ensembling***

The combination performed on the prediction of different models on different part of data set on your validation set. There are different types:

* **Averaging:**  averaging the predictions,
* **Weighted averaging:** averaging predictions by weighting models with higher quality,
* **Conditional:** take prediction for a range of target from one model, and the rest from other models
* **Bagging:** averaging slightly different independent version of the same model (random forest does this)
* **Boosting:** each model is built to explain the unexplained part of target for previous model(s). Gradient boosted trees do this.
* **Stacking:** training a model on the predicted value vs actual value of the validation target. Model is usually simpler. Called meta modelling. We divide data 50/50 between train/val.

**Tips on ensembling:**

* First ensemble several gradient boosted trees,
* 2-3 neural network,
* A few random forests,
* One linear regression
* A couple of K-NNs
* SVM with non-linear kernel
* Next layers have simpler algorithms,
* In next layers, feature engineering can be difference between predictions (y\_model1-y\_model\_2).