* 1. Supervised learning provides labels for training set data while unsupervised learning doesn’t.
  2. The training set is used to train the model on the data by minimizing its cost function. A validation set helps you pick the best model to fit your problem. The test set is a set of unseen data that you can ask your model to predict labels for, which will show you how it should perform in the real world.
  3. Whether it is representative of what post-production data would look like. (To help this along we commonly shuffle the dataset to get rid of any sorting).
  4. ***\*\*\*Write code to load a dataset and split it into X\_train, y\_train, X\_test, y\_test <Link>***

<https://colab.research.google.com/drive/1UDurTWNu9N4HN0ClCftjcjRWszugG8H9?usp=sharing>

* 1. i) Remove the feature(s) with missing entries

ii) Remove the instances with missing entries

iii) Add in a default value for instances with missing features (median from the dataset)

* 1. Split it into multiple binary features [using a One-Hot Encoder]
  2. Scale them (usually from 0 to 1)
  3. ***\*\*\*Write a pipeline to preprocess your features. Apply it to your train and test set. <Link>***

<https://colab.research.google.com/drive/1UDurTWNu9N4HN0ClCftjcjRWszugG8H9?usp=sharing>

* 1. Decision Tree
  2. A mix of experience with the model (to set ranges) and trying out many different ones using Grid/Randomized Search.
  3. ***\*\*\*Write code to find good hyperparameters for a given model. <Link>***

<https://colab.research.google.com/drive/1UDurTWNu9N4HN0ClCftjcjRWszugG8H9?usp=sharing>

* 1. Create a naive/simple model to set a baseline, and try to improve significantly from that result. If you do this and get a large paycheck, it’s performing well. :)
  2. F1 score, which is the harmonic mean of the precision and recall metrics.
  3. Use multiple binary classifiers, one for each label to determine the probability of it being that specific label. Then pick the highest probability of the bunch.
  4. You can check your F1 score for each label and weigh the average according to their amount of occurrences.
  5. ***\*\*\*Write code to evaluate your model.***

<https://colab.research.google.com/drive/1UDurTWNu9N4HN0ClCftjcjRWszugG8H9?usp=sharing>

* 1. i) Linear Regression: Reduce the feature weights allowed (include a higher cost in the cost function, making the slope flatter)

ii) SVM: Reduce C and/or gamma; Lower C values will prioritize a larger margin,

increasing the margin violations (and lowering the accuracy), which makes for a simpler

decision function. Lower gamma values will smooth the model by increasing each

datapoints’ influence.

(nonlinear) SVM: Reducing the polynomial degree, forcing it into a less accurate model.

iii) Decision Trees: Limit Max-depth hyperparameter, require a certain amount of

instances per node, etc. Letting the tree run its course and then pruning unproductive nodes is also an option.

* 1. Overfitting; regularization reduces the model’s complexity, preventing it from closely fitting all the data points in the training set.
  2. i) Overfitting: The model is very accurate on the training set but can’t generalize well.

ii) Underfitting: Inaccurate across the board; the model is too simple and doesn’t address

the complexities of the problem.

* 1. i) Overfitting: Model is too complex for the problem.

ii) Underfitting: Model is too simple for the problem.

* 1. i) Bias is caused by incorrect assumptions about the problem (thinking it to follow a specific pattern that isn’t true, etc.). This causes general inaccuracy in the model’s predictions.

ii) Variance is caused by a very free/complex model that allows overfitting the training

set; this reduces its ability to generalize to other data.

* 1. How does Linear Regression work?

\*\*\*Linear regression tries to find the ideal feature weights (and bias term) that minimizes the RMSE cost function.

* 1. Logistic regression is similar to linear regression; it takes a logistic of the weighted sum of the features, which is the probability that the instance belongs to a specific class. It is a binary classifier only. Logistic regression is trained using the log loss cost function, which puts large penalties on low probabilities for positive instances and high probabilities for negative instances.
  2. SVM’s split the different classes via the support vectors while providing as much wiggle room and as few margin violations as possible for predictions.
  3. Decision trees make iterative splits that branch out to divide the classes; the feature and values of where the node will split the samples is done based on which split will have the least impurity (class mixing). Once done, follow the splits based on your input feature values to find your predicted class.
  4. i) Using an equation (the “Normal Equation”) that determines the best fitting feature weights in one (rather large) computation

ii) Using gradient descent to get there in iterative fashioned steps.

The normal equation is much more plug & play; you don’t need to deal with hyperparameters or feature scaling. Gradient descent is, however, a faster algorithm in terms of algorithmic complexity with regard to a large number of features, and can possibly be used out-of-core depending on your specific version of GD.

* 1. They attempt to minimize the **w** and **b** values in the decision function; this is done using a quadratic programming solver.
  2. Using the CART algorithm; essentially it tries to find the best split at a feature and its threshold that have the purest (belong to a single class) children at each node. The algorithm iterates until it can’t find any more splits that reduce impurity or is forced to stop by some form of regularization.

* 1. i) Linear, Polynomial, & Gaussian RBF

ii) Linear kernel SVC is similar to a linear SVM; it linearly separates the classes (when possible) and uses that line as its decision function. Polynomial kernel does the same thing, but adds degrees of freedom according to your ‘degree’ hyperparameter, allowing for more complexity, with all the pros and cons that come with that. It may also make the data linearly separable if it wasn’t before. RBF kernel works based on similarity features, which we generate via landmarks.

iii) The kernel trick is the realization that actually adding more features to your dataset via transforming them using their dot product produces the same result as simply computing the dot product of the starting vectors to the polynomial degree of your choice. This prevents us from needing to transform and add those features, speeding up our training.

* 1. i) Put in hyperparameters (learning rate)

ii) Find immediate steepest slope downwards with regards to feature weights effect on the cost function

iii) Follow direction for distance according to the learning rate

iv) Repeat 2 & 3 equal to # of repetitions

* 1. i) Batch - Processes the entire training set at once; this is the slowest of the three but always converges to a global minimum (with a low enough learning rate and long enough training time), so very good for accuracy.

ii) Mini-Batch - Processes the training set in small chunks; middle of the road. Not as slow as Batch GD, and doesn’t have the high variance of stochastic GD, but loses out on some of their benefits.

iii) Stochastic - Processes the training set one instance at a time; very fast, and can be used out-of-core since it doesn’t need the whole training set in memory at once. Has accuracy issues because the instances have variance. Scales much better than the other two.

* 1. i) The learning rate determines the distance to travel (or amount to adjust) in the direction to minimize the cost function with respect to the feature weights.

ii) An overly large learning rate will prevent GD from converging to a minimum; it may bounce around the minimum back and forth, or possibly diverge and get further with each jump.

iii) When it is too small [and the cost function isn’t convex] it can get stuck in local minima thinking that it found the ideal feature weights. It can also simply run out of repetitions.

1. ***\*\*\*Write code to create an instance of each of the models we covered, find good***

***hyperparameters using a subset of your data, train it using cross-validation and find its performance, and evaluate it on your test set.***

<https://colab.research.google.com/drive/1UDurTWNu9N4HN0ClCftjcjRWszugG8H9?usp=sharing>