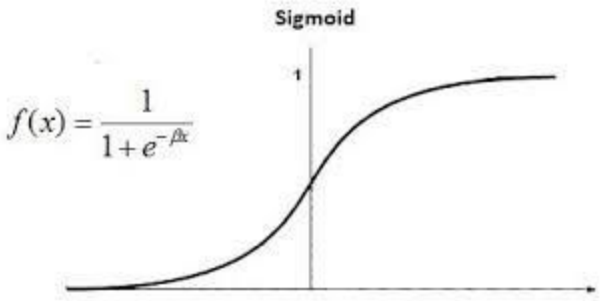
The Pros and Cons of each Modeling Technique

Figuring out which models to use in which situation can be challenging. To understand when to use which model, this requires understanding of the pros and cons of each Machine Learning technique.

Logistic Regression



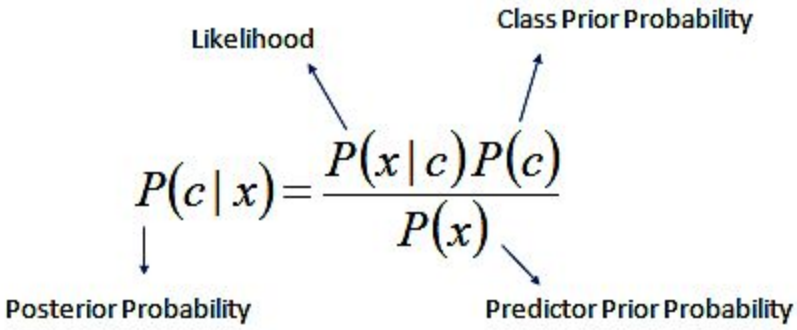
Overview:
Based on the principles of linear regression, we believe that there are certain coefficients or log-odds to lead to other 0 or 1 on a probability scale.

Pro:	Con:
A simple model to start just to test out the feature columns. Easily interpretable results with coefficients to see how the features are influencing the outcomes of 0 or 1.	When the feature set is increased, the model has a difficult time appropriating the coefficients. This could lead to overfitting and poor performance on your test set, especially when you deploy the model to the end-user.
Scalable solution on large rows of data, which means it is fast to train , deploy,and re-train. (**Scalable** means that you can increase the order of magnitude of the data. We can get results quickly, even if the workload increases.)	The model assumes a linear path (linear regression), which is rarely the case in the real world scenario. This means that the simplicity of the model will likely miss out on the complexity of certain datasets. Not every feature is going to lead to log odds to 0 or 1 in such a neat fashion.

Conclusion:

Typically, this is not a model to use in production since there are much better options for modeling. This is, however, a good, fast, and simple model to start in the modeling process simply to test out the features in a dataset and to get a sense of what is important quickly. You can also use this model to "debug" your dataset since you can quickly realize that there are missing gaps or null values without dealing with the complexities of higher modeling techniques.

Naive Bayes



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Overview:
With Naive Bayes, there is a naive assumption that each conditional variable is independent of one another. With that assumption, we can multiply the variables and produce a joint probability score that is towards 1 or 0.

Pro:	Cons:
Based on the naive assumption of independence, this is a fast, scalable solution to train, deploy, and retrain a statistical model.	Long complex multiplication of probabilities will eventually widdle down even the most positive classification. Not good for many features such as 50.
Great for text classification. A group of words used can either lead to a positive sentiment or negative sentiment. This was the ideal model choice for spam filters, spam/ham for many years.	One value that is heavily weighted in the probability sequence can drastically change the outcomes of the prediction output. Making other variables not important. For example, one very negative word can bring the entire sentiment of the text corpus. Structure does not look at the word use case like a Neural Network.

Suitable for large datasets due to the relative simplicity of the math.	Not able to be used as a regressor. Most models can be used as both with some slight changes.
The conditional probabilities give a quantifiable score.	If a categorical variable has a category (in the test data set), which was not observed in the training data set, then the model will assign a 0 (zero) probability and will be unable to make a prediction. To solve this, there is a method called Add 1 smoothing just to have something to divide with.

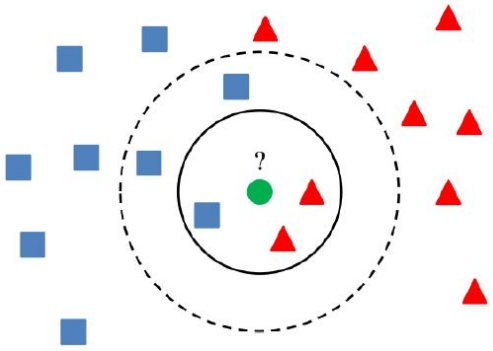
Example:
https://www.geeksforgeeks.org/naive-bayes-classifiers/

Conclusion:
This model is typically used for fast text classification with favorable results with few downsides.

Note:

As we move up towards more complex models, the models move away from parametric models (equations) to more nonparametric(algorithmic) models. This is also a shift from statistical techniques to more computer science techniques.

KNN

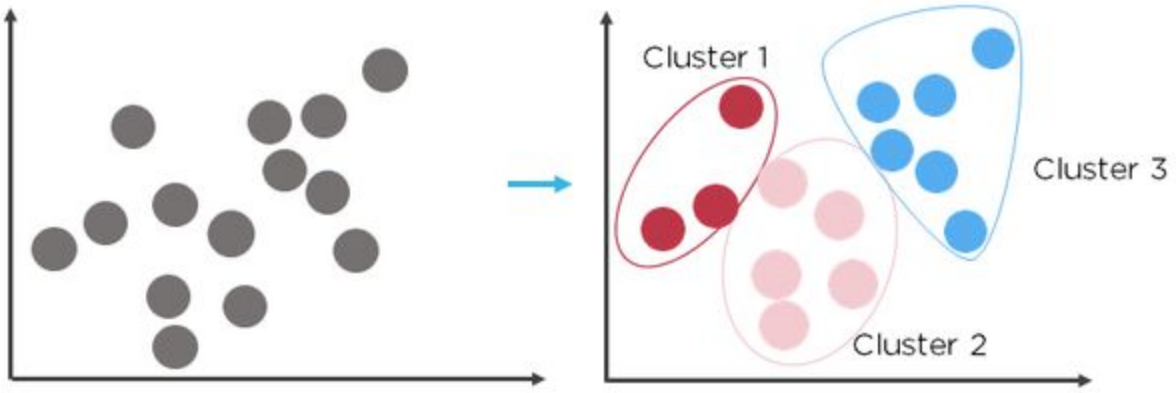


Overview:
The model exhaustively uses distance metrics (Euclidean Distance) and classifies your new datapoint by its closest neighbors. **Exhaustive** means going through all of the combinations of distances between all points in relation to each other.

Pros:	Cons:

The formula is simple to apply where you are using distance metrics between data points. You set K neighbors to find out if 3 out of 5 data points point to one class then, we can say that it is that class.	Does poorly on high dimensional data or lots of columns. We would need to transform the dataset with dimensionality reduction to plot the dataset in the 2D plane to use distance metrics to say which neighbors are closest to this observed, therefore you are that class.
The model is not necessarily trained, but the distances can be saved and reused to find the nearest neighbor for new observances of data. Easy Non parametric Model. Not bounded by a strict formula. This is a "**lazy** learner", which means it does not have a training phase.	Computationally, it can be time-consuming to run exhaustively through the entire dataset to get the metrics and find the nearest neighbors in large row-wise datasets.

Conclusion:
A KNN can be a great model for a recommender, where you can bundle similar items by it's nearest neighbor. It is also great to use this model after using unsupervised clustering by using the cluster assignments as labels.



The labels will not be completely accurate, but it is a quick way to turn an unsupervised problem into a supervised problem without too much effort.

Levels of Tree-based modeling

Decision Tree → Random Forest → Boosted Models

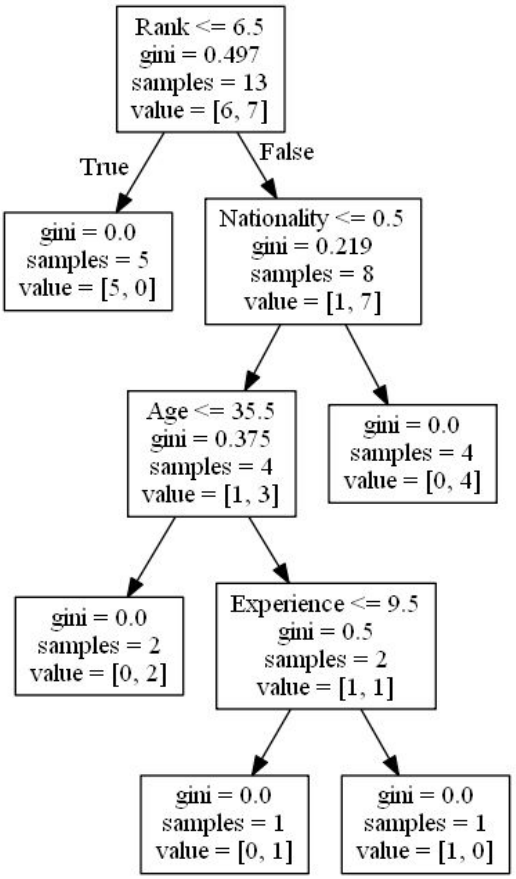
Decision Tree

Computer science concepts:
Binary Decision Tree
Recursive Partitioning

Example

Based on someone's age, rank, experience, and nationality, will that person go to the comedy show or not?

```
Nationality = {'UK': 0, 'USA': 1, 'N': 2}
```

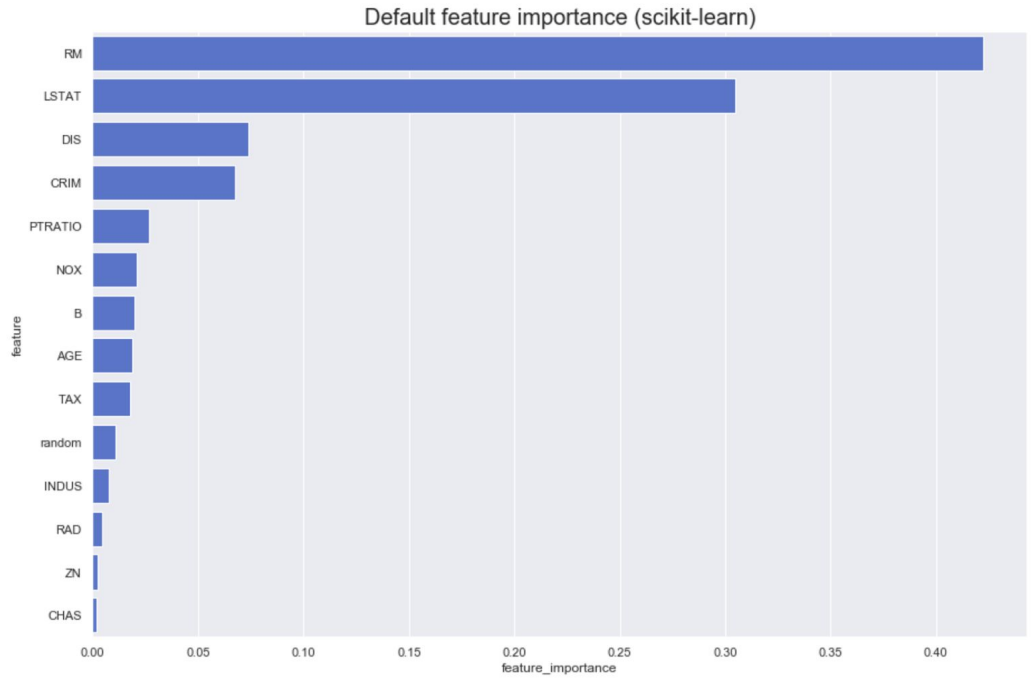


Notice the Gini (information gain) output, as you go down the decision tree, Gini decreases until it is 0, which means there was a decision made. This is a recursive process until Gini runs out. The bigger the Gini value that was used at the tree node, the bigger the split, the more important the feature. The depth of the tree is decided on its own. The more complicated the decision(more features), the bigger the decision tree. You can decide to place a limit on the depth so the model will not get too big.

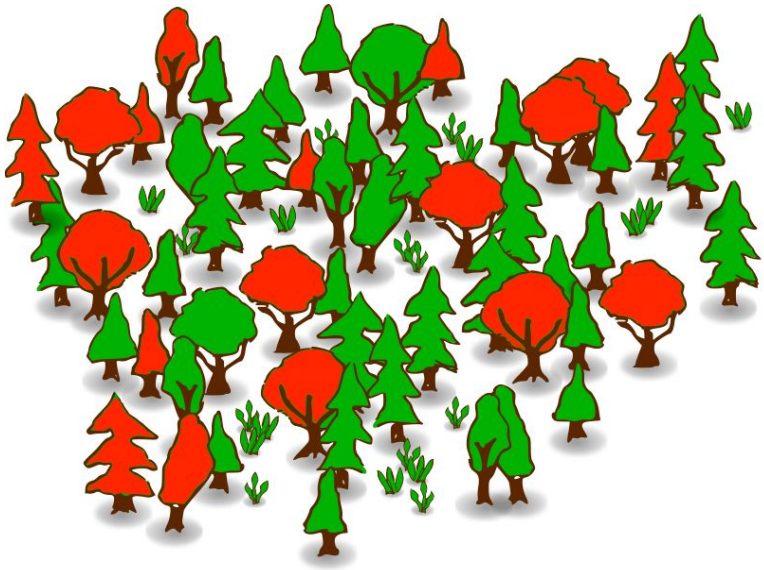
Pros:	Cons:

Good for small datasets, a few thousand rows, and 5-10 columns. A Random Forest Classifier with its ability to bootstrap aggregate is not helpful	Variables not explainable like Logistic Regression.
Algorithmically decides how many decisions to make, without having to “instruct” it to, through recursion of a binary tree. This makes the model very flexible and adaptable to many datasets.	Not good at big datasets with many rows and columns. A Random Forest would be better since the model will sample and create ‘mini’ decision trees. Making a decision on just one tree on a complicated dataset then becomes not ideal. At that point, it’s better to take subsets of the data instead.
Feature Importance Chart can show you the most predictive features. It may not show you whether the feature is influencing the model to 1 or 0, but the feature importance chart can show you that the feature is impactful to the predictability of the model performance.	

Example of a feature importance:



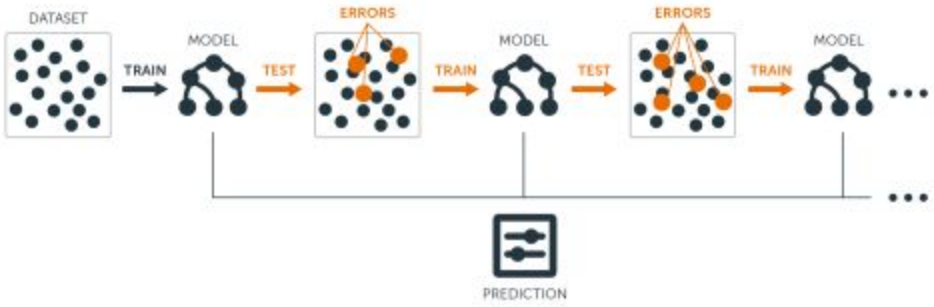
Random Forest Classifier



This is the next step in the process for tree based modeling

Pros:	Cons:
This is an “ensemble method” which means multiple trees to make a decision as a whole, which creates a stronger answer by independent consensus.	Does not do well on the imbalanced data. If there are not many samples for a class, it is difficult to create a variety of decision trees for your ensemble or “forest”.
Works well with models on large datasets row and column-wise, since it will be able to create a variety of different trees from sampling with replacement.	Prone to overfitting, the decision trees learn the dataset too well. There are techniques to “prune” the tree at the start or the end of the decision tree.
Same benefit of the feature importance chart to understand the most predictive features.	If there are few rows and columns, then there is not much to sample on when it comes to sampling with replacement
	Adding more N estimators (trees) over time does not increase its performance at a certain point. Random sampling with replacement can sometimes produce “weak learners”.

Boosted Models

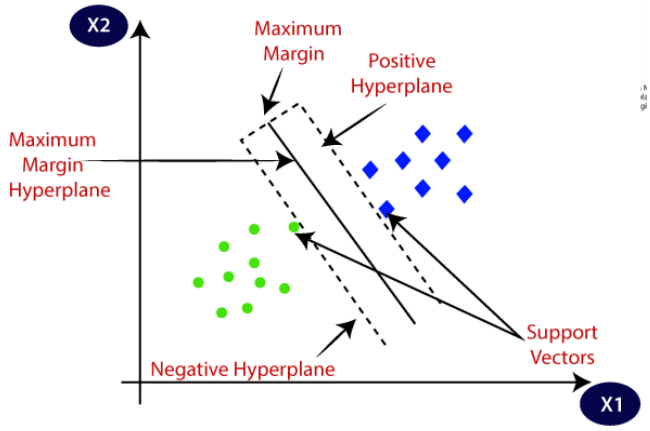


Overview:
Since Random Forest generates “weak learners” depending on the sample, boosted models were created to resolve this problem. There are different variations such as Gradient Boosting, Adaptive Boosting, and XG Boosting, but the concept is identifying the weak learners by assigning a weight (coefficient) and then “fix” them by combining them together in an iterative fashion to improve failed predictions in classification.

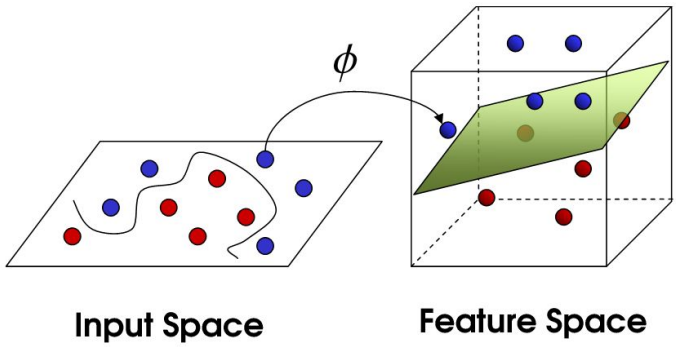
Pros:	Cons:
Give “weak learners” a chance to improve using the methods like the Gradient.	Finding “weak learners” can be slow and tedious for the model algorithm.
There are several other ways to “boost” the model as well such as ADA , XGBoost.	May not result in significant improvement for the model outcome. Certain “weak learners” will be fixed, but some won’t no matter how many passes through the dataset(iterations). The outcome will stay.
Popular choice for data science competitions.	

Conclusion:
This is a popular choice if your Random Forest Model is giving you good results, you and want to see if you can get even better results with a Boosted Model. There is also XGBoost available in BQML.

Support Vector Classifier

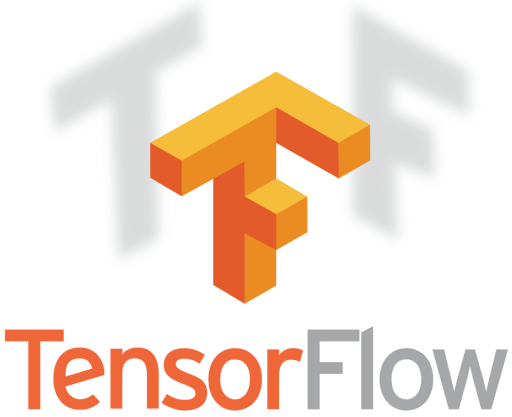


Overview:
Support Vector Machines create a separation between two classes with a hyperplane to bisect the data, with support vectors to create a margin. This requires the “kernel trick” to raise the dimensions 1 dimension higher to create this separation.



Pros:	Cons:
Good for imbalanced datasets, high effective for bio or medical datasets, where the classes are typically highly imbalanced.	Slow to train at Large datasets. This is because doing a hyperplane on a large dataset is very computationally intensive to do the vector transformations necessary.
Different kernels are available to create separation between classes.	For classes that are closely related to each other, it is difficult to create a clear separation for classification, despite trying different Kernels.
Great for datasets with a clear separation between datasets.	

Tensorflow (Neural Networks)



Pro:	Cons:
Best suited for image classification such as X-rays and identifying faces. It is also good for text classification if the corpus is big enough to support it.	Tabular data such as patient health is better suited for traditional machine learning methods. The complex nature of layers of a neural network does not typically add to any improvement in model metrics. If it is, then the gain is negligible.
Highly accurate when it comes to classifying images because it can learn the nuances of the images at the pixel level. Also, the many passes through the dataset, helps the model relearn.	Tensorflow can do regression, but the model structure is overly complicated for that task. It is however good for forecasting and predicting the next word in a sentence, or speech patterns.
Lots of parameters to tune and use to regularize the model to prevent overfitting.	Can be difficult to move forward in production due to the complexity and speed of the prediction. But GCP has tools to train and deploy, host, and retrain the model in the cloud if necessary with TFX .
Keras and TF 2.0 is simplifying the complexity of writing code for Neural Networks. The focus is more about knowing which type of Neural Network to use and when.	Sometimes the model does not converge no matter how long it's being trained. Also, most Tensorflow projects would require a GPU to speed up training, which can be costly.

Conclusion:
If you have images, it's best to go straight to Tensorflow/Keras. Although you can use traditional machine learning techniques for image classification, the accuracy is not there because Tensorflow can examine the image at a pixel level and also pass through many iterations to really learn about the image dataset. We can get a better sense of the features that separate a cat or a dog such as a nose,ears,paws, etc.

Comment on Regression:

Most models can be used as a regression model such as:

- Support Vector Regressor (SVR)
- K Nearest Neighbors Regressor
- Random Forest Regressor
- Gradient Boosted Regressor
- Tensorflow (Not Advised, Overkill)

Models that are not able to be Regression models (classification only):

- Naive Bayes
- Logistic Regression (Classification Typically)

Although the main focus is classification, we want to highlight the possibility of using the models as more complex regression techniques. We can explain further if you need us to.

Final Thoughts

Typically, there is a tradeoff between speed and accuracy. The more complex models are more accurate but more difficult to explain and use. The difficulties could be model maintenance, retraining, expensive to host, and slow to get predictions.

The simpler models are easier to explain, fast, easy to retrain to adjust to new massive incoming data, but the accuracy is not quite there. In certain situations, a moderate model performance is "good enough". If you are not worried about speed or scalability, and want to focus more on getting a good accurate model, then this is not a problem.

The parametric models are more statistical and bound by their model formulae such as Naive Bayes and Logistic Regression. The more advanced models become more "BlackBox" where you have an accurate output, but can't quite explain what led to that outcome. Fortunately, there are tools such as Tensorboard to peek under the hood of your Neural Network.

But whichever model you decide to use, there plenty of tools to collect,train, host the model, and scale it to your users in GCP.

Even in BQML, you can prep in BigQuery, train a model, validate, and hose the model in the cloud as a BQ Table.

The most important part of the modeling is the data prep to have clean, useful, and predictive features(attributes) for the model. Without it, no matter what the model choice , the model will not give you good predictions.

This is like pouring contaminated gasoline with gunks(missing/bad data) in your performance race car (Tensorflow).

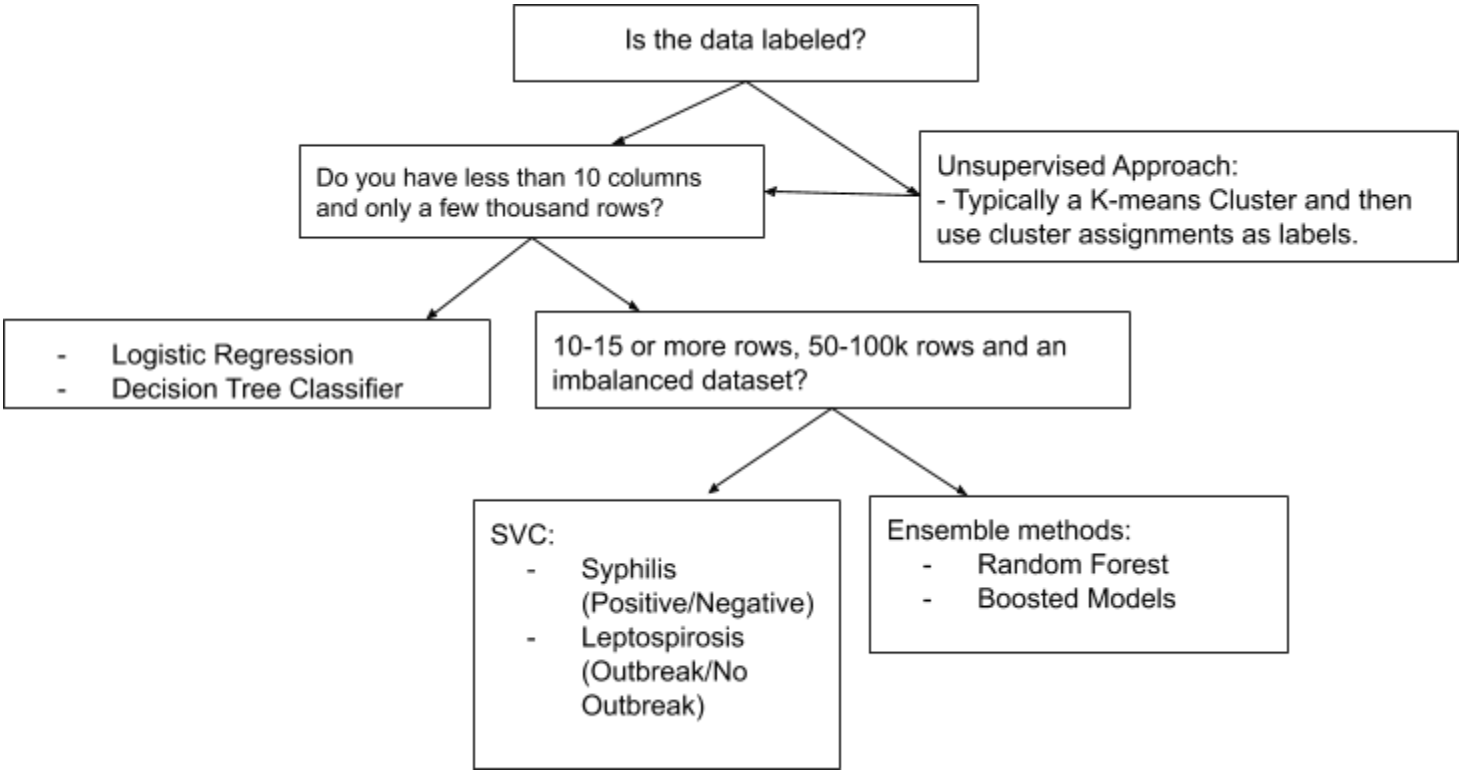
Lastly, doing well with train/test split is great, but the most important part of modeling is how it does “in the wild”. Spam for example changes, which means the model would need to be updated over time to handle “model drift”.

Also, if model predictions did fantastic on train/test, but failed in the real world, this usually means that the model is overfit due by not having a representative sample of the population for your classification model. Adding more regularization parameters on your model will most likely not fix a misrepresented sample size. This would mean several iterations resampling ,retrain, retest and deploy to get it right.

This is the part that is more the “art” than “science”.

These are guidelines and you will know better by doing and applying these concepts to your own dataset.

Recommended ML choices for Mantech use cases:
(The assumption is that the data is tabular)



SVC would be a good model for the 2 use cases identified as classification because we assume that these are highly imbalanced datasets with relatively less rows (under 50K). We also assume very few people will test positive for Syphilis and people will be healthy, creating the imbalance. Because of this, it has been popular with the medical , biology, chemistry community for years.