Pros and cons of ML algorithms

SVM

Pros:
1. Performs well in higher dimensions
2. Best algorithm when classes are separable
3. Outliers have less impact
4. Suited for extreme case binary classification

Cons:
1. Slow: for larger dataset, it requires a large amount of time to process
2. Poor performance with overlapped classes.
3. Selecting appropriate hyperparameters is important.
4. Selecting the appropriate kernel function can be tricky.

Naive Bayes

Pros:
1. Real time prediction.
2. Scalable with large datasets
3. Insensitive to irrelevant features
4. Multi class prediction.
5. Good performance

Cons:
1. Independence of features doesn't hold.
2. Bad estimator: Probability outputs from predict_proba are not to be taken too seriously.
3. Training data should represent population well.

Logistic Regression

Pros : ① Simple to implement
② Effective
③ feature scaling not needed
④ Tuning of hyperparameters not needed.

Cons : ① Poor performance on non-linear data (e.g., image data)
② Poor performance with irrelevant and high correlated features
③ Not very powerful algorithm.

Random Forest

Pros : ① Reduced risk of overfitting
② Good performance on imbalanced datasets.
③ Can handle huge amounts of data
④ Good handling of missing data
⑤ Little impact of outliers.
⑥ Useful to extract feature importance.

Cons : ① Time consuming process
② Requires more resources
③ Appears as a black box

Decision Trees

Pros:
1. Normalization or scaling of data not needed.
2. Handles missing values.
3. Easy to explain
4. Easy visualization
5. Automatic feature selection.

Cons:
1. Prone to overfitting
2. Sensitive to changes in data.
3. Higher time required to train.

KNN

Pros:
1. Simple to understand and implement
2. No assumption about data (e.g. in case of linear regression we assume dependent variable and independent variables are linearly related, in Naive Bayes we assume features are independent of each other etc.)
3. Constantly evolving model.
4. Multi-class problems can be solved
5. One hyper-parameter to tune.
6. No training involved ("lazy"). New training examples can be added easily.

Cons : ① Slow for large datasets
② Doesn't work very well on datasets with large number of features
③ Scaling of data is required
④ Doesn't work well on imbalanced data.
⑤ Sensitive to outliers
⑥ Cannot deal well with missing values.

- Expensive & slow: $O(mnd)$, m = # examples, d = # dimension
- To determine the nearest neighbor of a new point x, must compute the distance to all m training examples. Runtime performance is slow, but can be improved
  - Pre-sort training examples into fast data structures
  - Compute only an approximate distance
  - Remove redundant data (condensing).

XGBoost

Pros : ① Less feature engineering required (no need for scaling, normalizing data, can also handle missing values well)
② feature importance can be found
③ fast to interpret
④ Outliers have minimal impact
⑤ Handles large datasets.
⑥ Good execution speed
⑦ Good model performance
⑧ Less prone to overfitting.

Cons :
① Difficult interpretation, visualization tough
② Harder to tune as there are many hyperparameters.

K-Means

Pros:
1. Relatively simple to implement
2. Scales to large data sets
3. Guarantees convergence
4. Can warm start the position of centroids
5. Easily adapts to new examples
6. Generalizes to clusters of different shapes and sizes, such as elliptical clusters

ons:
1. Choosing k manually
2. Being dependent on initial values
3. Clustering data of varying sizes and density.
4. Clustering outliers.
5. Scaling with number of dimensions.