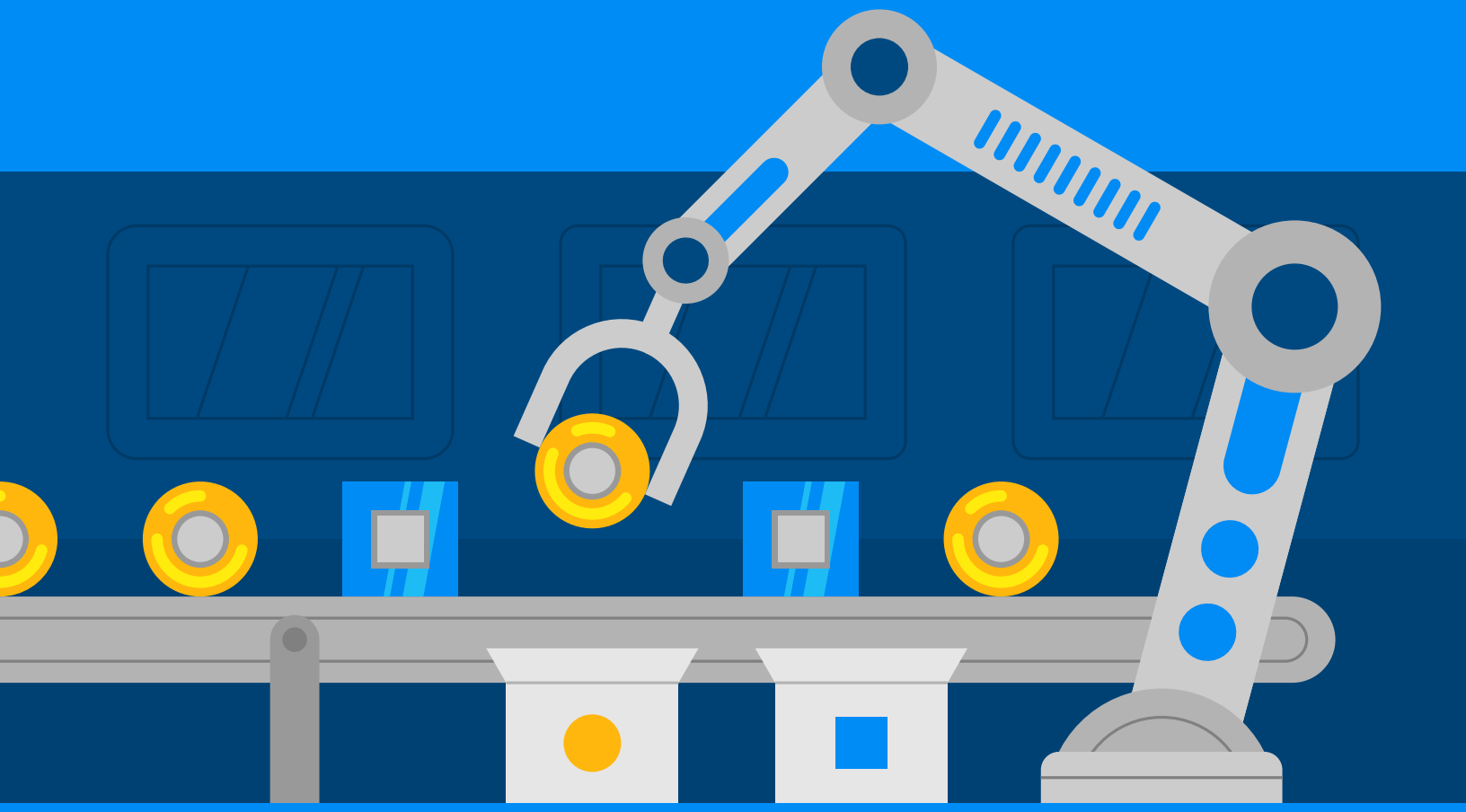
MACHINE LEARNING CLASSIFICATION BOOTCAMP CHEATSHEET



SUPER DATASCIENCE
MAKING THE COMPLEX SIMPLE

1. CONFUSION MATRIX/ CLASSIFICATION REPORT

1.A Confusion matrix concept

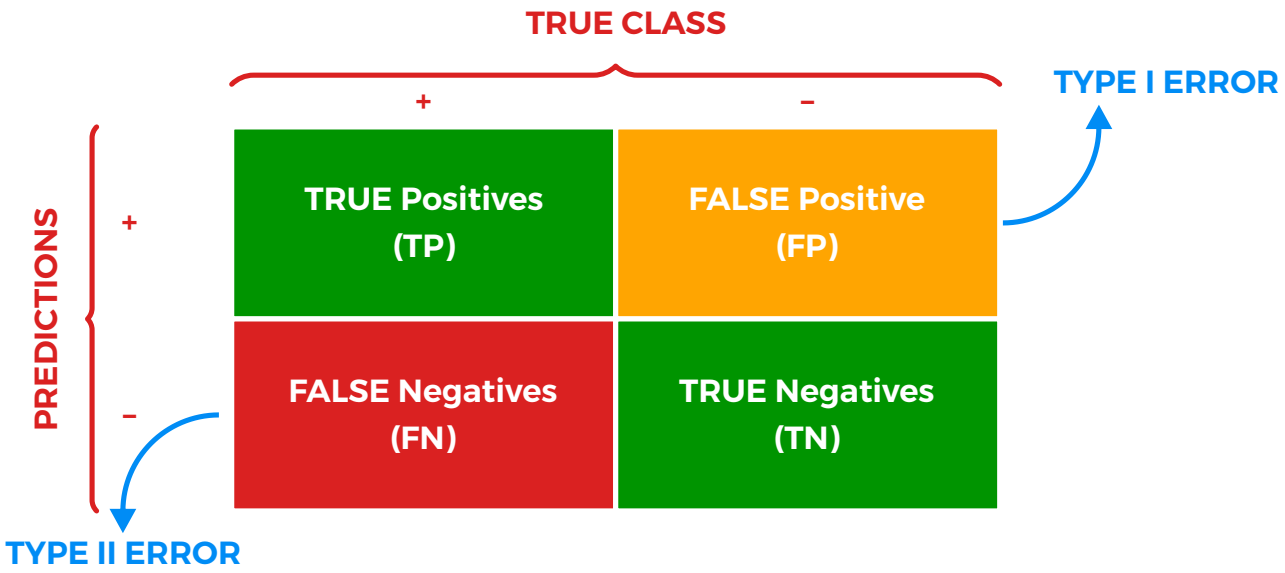
A confusion matrix is used to describe the performance of a classification model:

True positives (TP): cases when classifier predicted TRUE (they have the disease), and correct class was TRUE (patient has disease).

True negatives (TN): cases when model predicted FALSE (no disease), and correct class was FALSE (patient do not have disease).

False positives (FP) (Type I error): classifier predicted TRUE, but correct class was FALSE (patient did not have disease).

False negatives (FN) (Type II error): classifier predicted FALSE (patient do not have disease), but they actually do have the disease



- Classification Accuracy = (TP+TN) / (TP + TN + FP + FN)
- Misclassification rate (Error Rate) = (FP + FN) / (TP + TN + FP + FN)
- Precision = TP/Total TRUE Predictions = TP/ (TP+FP) (When model predicted TRUE class, how often was it right?)
- Recall = TP/ Actual TRUE = TP/ (TP+FN) (when the class was actually TRUE, how often did the classifier get it right?)

1.B Confusion matrix in sklearn

```
from sklearn.metrics import classification_report, confusion_matrix
y_predict_test = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_predict_test)
sns.heatmap(cm, annot=True)
```

1.C Classification report

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

2. DIVIDE DATASET INTO TRAINING AND TESTING

2.A Concept

Data set is generally divided into 75% for training and 25% for testing.

Training set: used for model training.

Testing set: used for testing trained model. Make sure that testing dataset has never been seen by the trained model before.

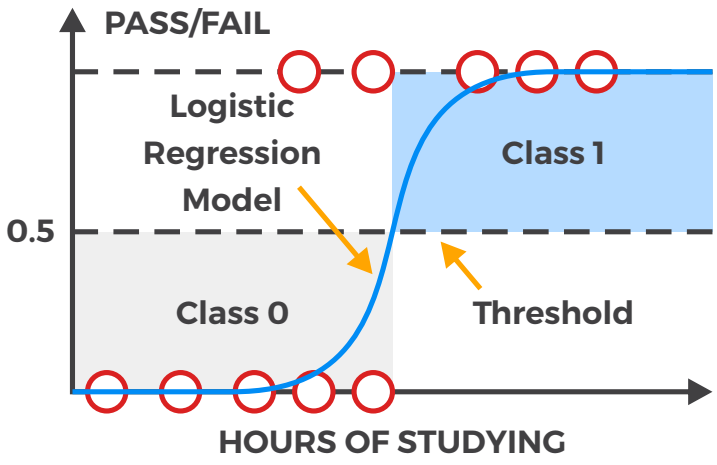
2.B Dividing dataset in sklearn

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2, random_state=0)
```

3. LOGISTIC REGRESSION

3.A Logistic regression concept

- Logistic regression predicts binary outputs with two possible values labeled "0" or "1"
- Logistic model output can be one of two classes: pass/fail, win/lose, healthy/sick
- Logistic regression algorithm works by implementing a linear equation first with independent predictors to predict a value.
- This value is then converted into a probability that could range from 0 to 1.



Step #1: Start with a Linear equation:

$$y = b_0 + b_1 * x$$

Step #2: Apply Sigmoid function to get probability:

$$P(x) = \text{sigmoid}(y)$$

$$P(x) = \frac{1}{1 + e^{-y}}$$

$$P(x) = \frac{1}{1 + e^{-(b0 + b1 * x)}}$$

3.B Logistic regression in sklearn

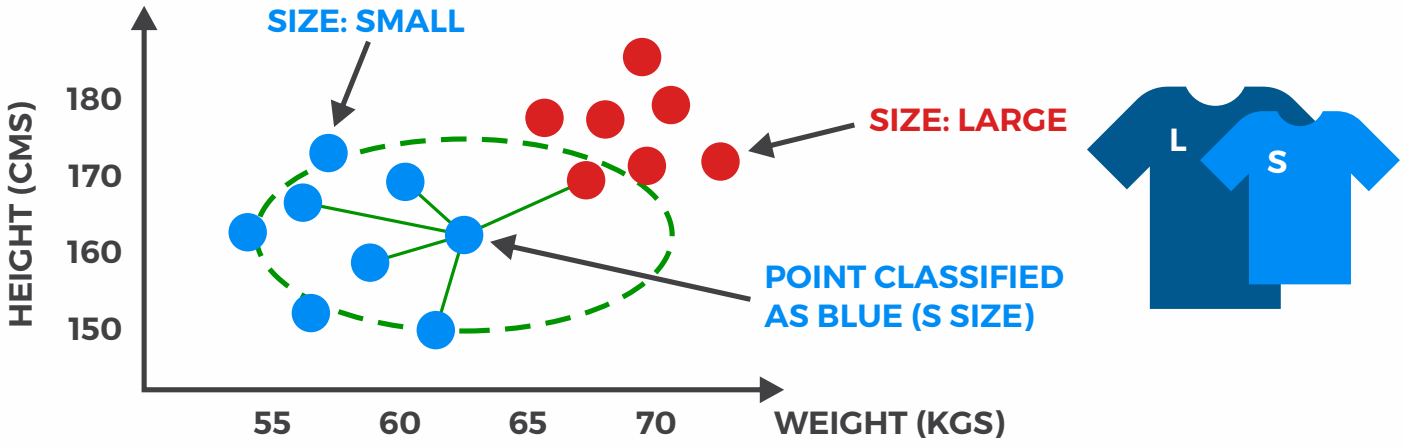
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

4. K-NEAREST NEIGHBORS

4.A K-nearest neighbors concpet

- K-nearest neighbor algorithm (KNN) is a classification algorithm that works by finding the most similar data points in the training data, and attempt to make an educated guess based on their classifications.
- KNN Algorithm steps:

1. Select a value for k (e.g.: 1, 2, 3, 10..)
2. Calculate the Euclidian distance between the point to be classified and every other point in the training data-set
3. Pick the k closest data points (points with the k smallest distances)
4. Run a majority vote among selected data points, the dominating classification is the winner! Point is classified based on the dominant class.
5. Repeat!



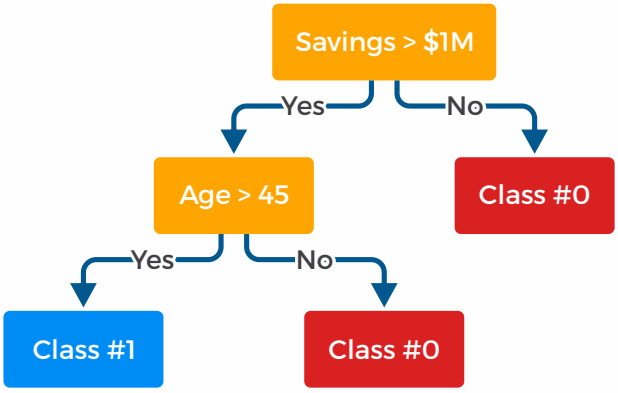
4.B K-nearest neighbors in sklearn

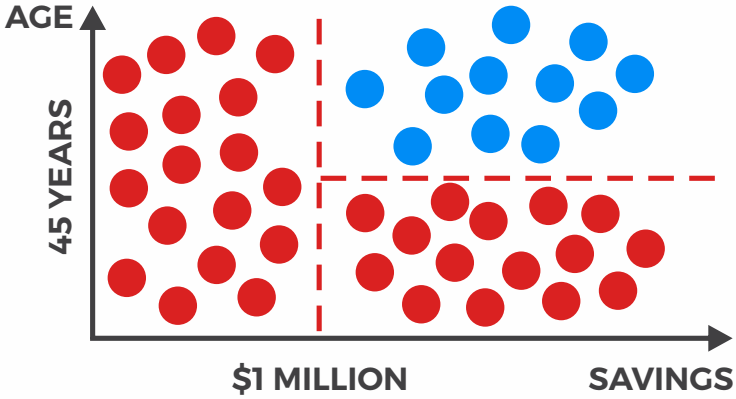
```
from sklearn.linear_model import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

5. DECISION TREES CLASSIFIER

5.A Decision trees concept

- Decision Trees are supervised machine learning technique where the data is split according to a certain condition/parameter.
- The tree consists of decision nodes and leaves.
- Leaves are the decisions or the final outcomes.
- Decision nodes are where the data is split based on a certain attribute.
- Objective is to minimize the entropy which provides the optimum split.





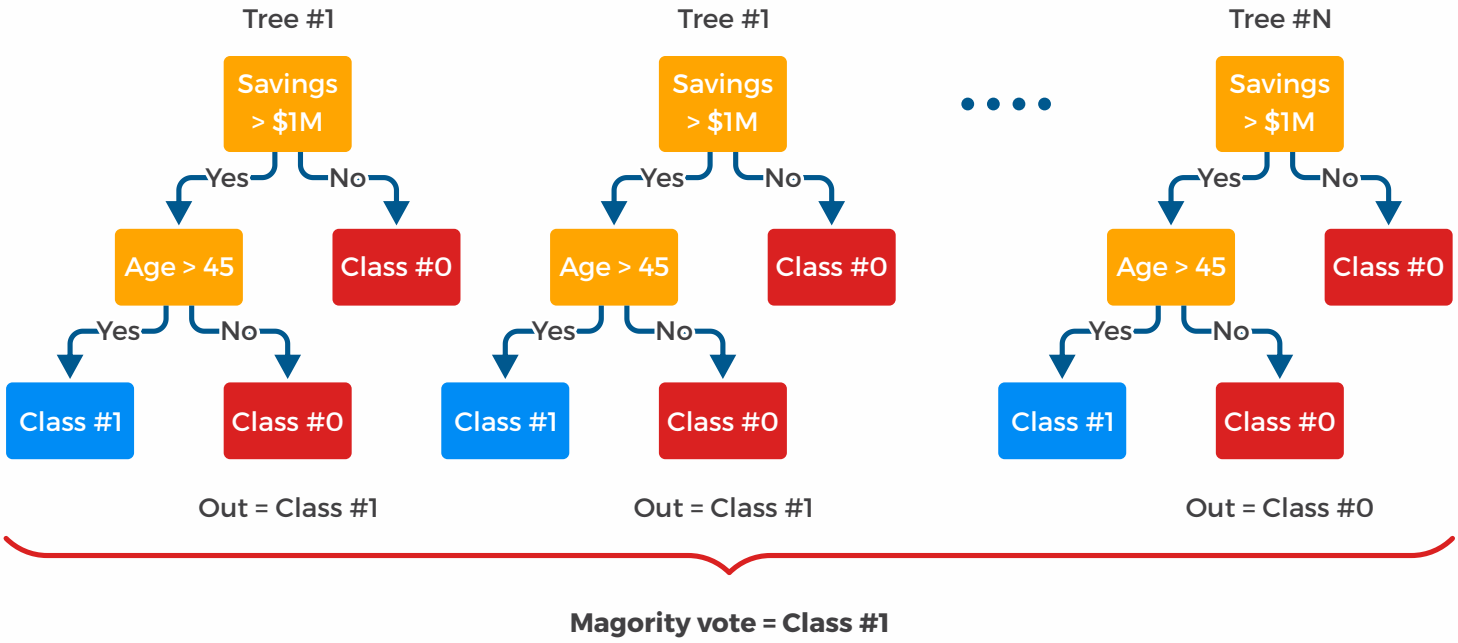
5.B Decision trees in sklearn

```
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
```

6. RANDOM FOREST CLASSIFIER

6.A Random forest concept

- Random Forest Classifier is a type of ensemble algorithm.
- It creates a set of decision trees from randomly selected subset of training set.
- It then combines votes from different decision trees to decide the final class of the test object.
- It overcomes the issues with single decision trees by reducing the effect of noise.
- Overcomes overfitting problem by taking average of all the predictions, cancelling out biases.



6.B Random forest in sklearn

```
from sklearn.ensemble import RandomForestClassifier
RandomForest = RandomForestClassifier(n_estimators=250)
RandomForest.fit(X_train, y_train)
```

7. NAIVE BAYES

7.A Naive bayes concept

Naive Bayes is a classification technique based on Bayes' Theorem.

$$P(\text{Retire}|X) = \frac{P(X|\text{Retire}) * P(\text{Retire})}{P(X)}$$

LIKEHOOD | PRIOR PROBABILITY OF RETIRING | MARGINA LIKEHOOD

X: New Customer's features; age and savings
P(Retire|X): probability of customer retiring given his/her features, such as age and savings
P(Retire): prior probability of retiring, without any prior knowledge
P(X|Retire): likelihood
P(X): marginal likelihood, the probability of any point added lies into the circle

7.B Naive bayes in sklearn

```
from sklearn.naive_bayes import GaussianNB
NB_classifier = GaussianNB()
NB_classifier.fit(X_train, y_train)
```

8. SUPPORT VECTOR MACHINES

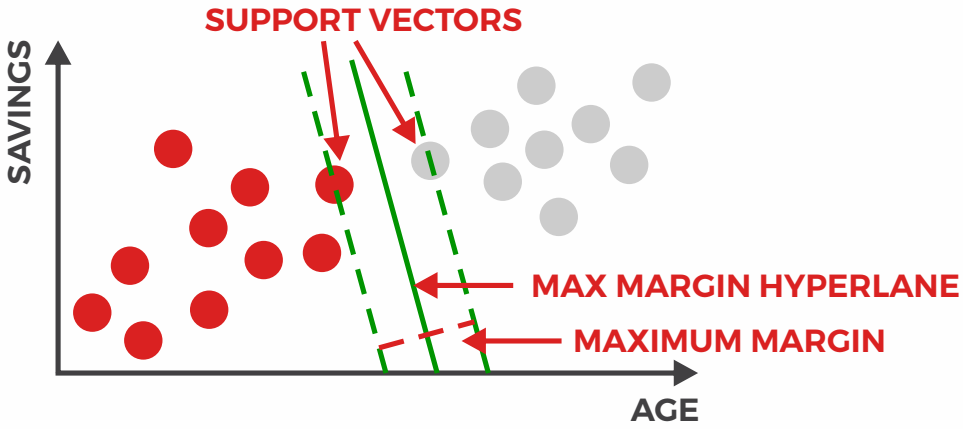
8.A Support vector machines concept

Gamma parameter: controls how far the influence of a single training set reaches

- Large gamma: close reach (closer data points have high weight)
- Small gamma: far reach (more generalized solution)

Gamma parameter: controls how far the influence of a single training set reaches

- Small C (loose) makes cost (penalty) of misclassification low (soft margin)
- Large C (strict) makes cost of misclassification high (hard margin), forcing the model to explain input data stricter and potentially over fit.



8.B Support vector machines in sklearn

```
from sklearn.svm import SVC
svc_model = SVC()
svc_model.fit(X_train, y_train)
```

8.C Parameters optimization in sklearn

```
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf']}
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=4)
grid.fit(X_train_scaled,y_train)
grid.best_params_
grid.best_estimator
```