17/2/2024

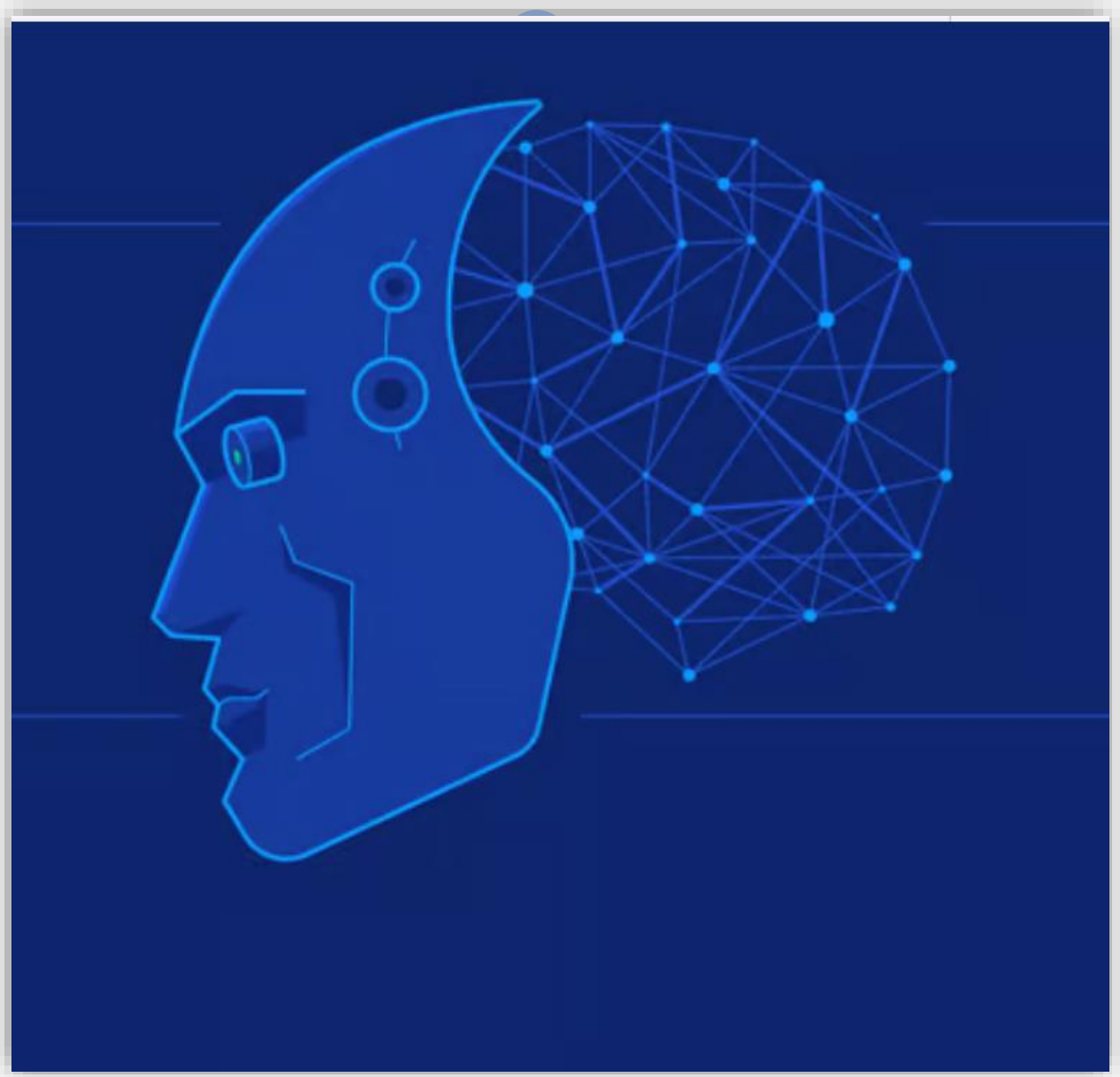
Predicting heart disease

February 17, 2024

Khaledgama4@gmail.com

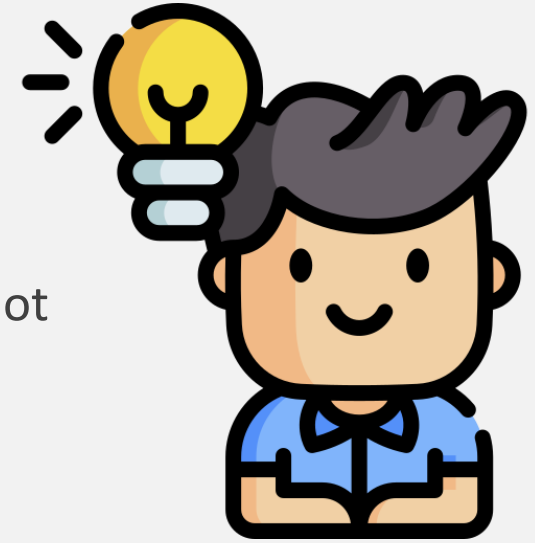
Agenda

- Problem definition
- Data
- Modelling
- Evaluation
- Tuning a model (improving it)



Problem Definition

Given clinical parameters about a patient, can we predict whether or not they have heart disease?



Data interpretation

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Check Nulls & Datatypes

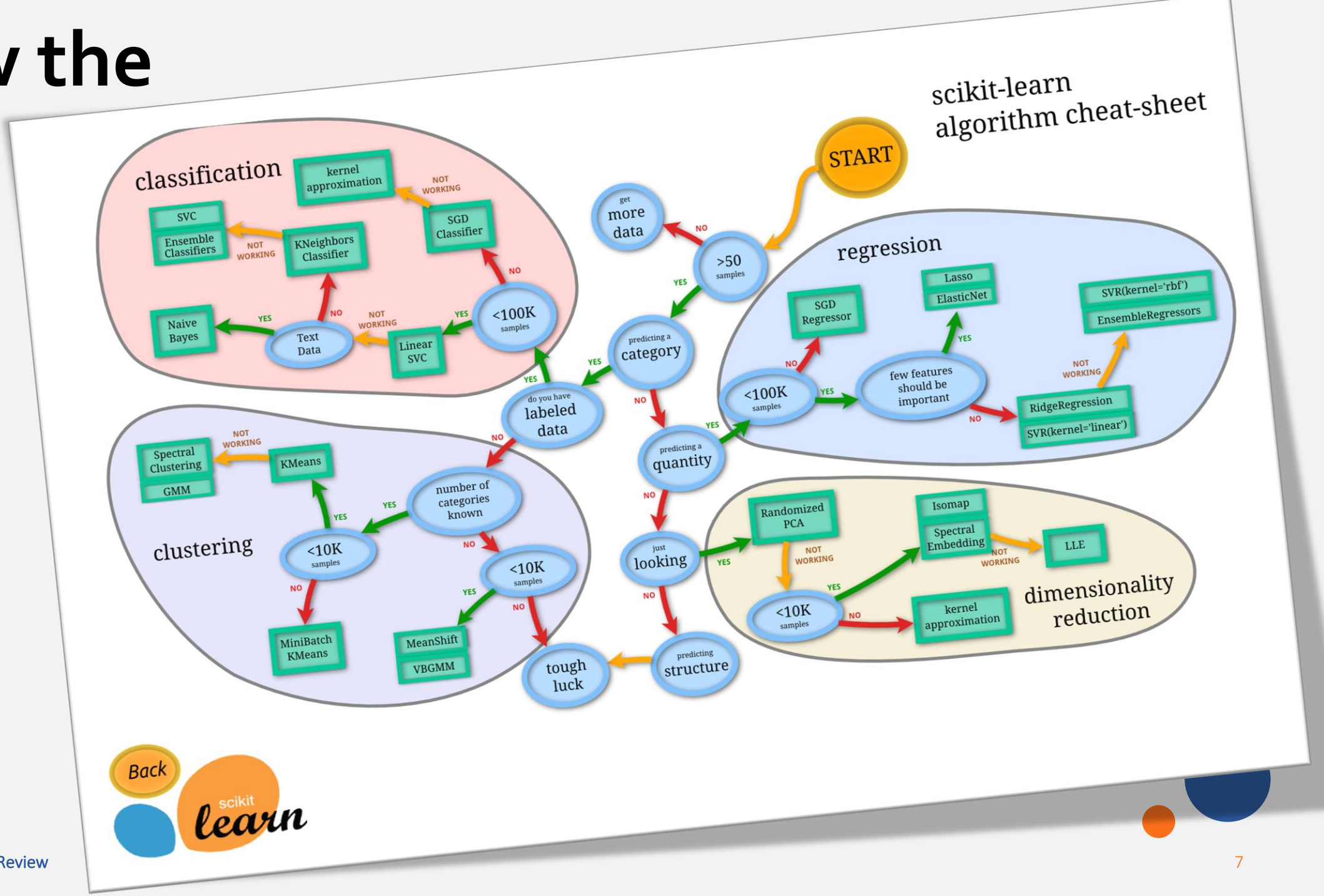
```
In [5]: heart_disease.isna().sum()

Out[5]: age         0
        sex         0
        cp          0
        trestbps    0
        chol        0
        fbs         0
        restecg     0
        thalach     0
        exang       0
        oldpeak     0
        slope       0
        ca          0
        thal        0
        target      0
        dtype: int64
```

```
In [4]: # Check if there were missing values or categorical data
        heart_disease.dtypes

Out[4]: age           int64
        sex           int64
        cp            int64
        trestbps      int64
        chol          int64
        fbs           int64
        restecg       int64
        thalach       int64
        exang         int64
        oldpeak     float64
        slope         int64
        ca            int64
        thal          int64
        target        int64
        dtype: object
```

Choose the right algorithm!

How can I choose the right estimator for my case !!!!

Follow the map!



LinearSVC estimator!

In [27]:
```
# Choose the algorithm & estimator
# Follow the sci-ket map
# Choose linear svc (support vector classifier)
np.random.seed(seed=0)
from sklearn.svm import LinearSVC
clf = LinearSVC()

# instantiate and fit the model
clf.fit(x_train,y_train)

# check the model score !!
clf.score(x_test,y_test)
```

```
C:\Users\DELL\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will
change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\DELL\anaconda3\Lib\site-packages\sklearn\svm\_base.py:1242: ConvergenceWarning: Liblinear failed to converg
e, increase the number of iterations.
  warnings.warn(
```

Out[27]: 0.45901639344262296

LinearSVC estimator!

In [27]:
```
# Choose the algorithm & estimator
# Follow the sci-ket map
# Choose linear svc (support vector classifier)
np.random.seed(seed=0)
from sklearn.svm import LinearSVC
clf = LinearSVC()

# instantiate and fit the model
clf.fit(x_train,y_train)

# check the model score !!
clf.score(x_test,y_test)
```

```
C:\Users\DELL\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will
change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\DELL\anaconda3\Lib\site-packages\sklearn\svm\_base.py:1242: ConvergenceWarning: Liblinear failed to converg
e, increase the number of iterations.
  warnings.warn(
```

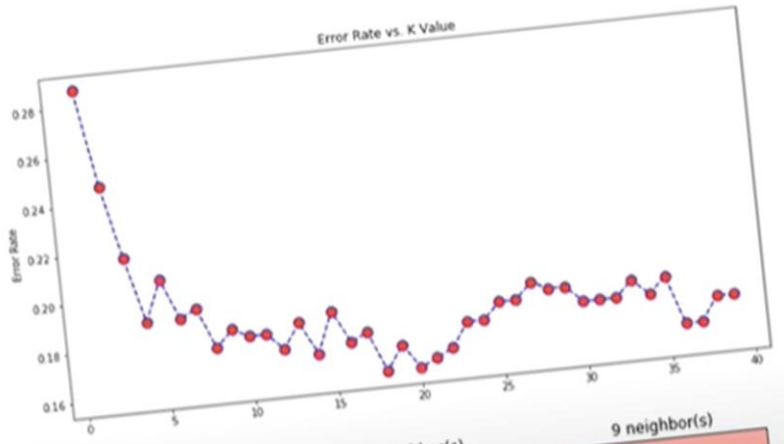
Out[27]: 0.45901639344262296

KNN

Hyperparameter Tuning (Choosing 'K')

euclidean- manhattan- minkowski

- The number of nearest neighbors to compare a record to, K, is determined by **how well the algorithm performs on training data**, using different values for K.
- Choosing **K=1**, or **K = even number** (2 for example) is very **noisy** and **sensitive to outliers**.
- Start with **(K=1)**, then try **(K=K+2)**.
- Choose the value of 'K' for which the **error rate** is **minimum**.
- **Decision boundary** becomes **smoother** as 'K' **increases** (less overfitting).
- Choosing **the distance measure** will affect the final results.



KNN estimator!

In [28]:
```
# Choose KNN (K neighbour classifier)
np.random.seed(seed=42)
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier()

# instantiate and fit the model
clf.fit(x_train,y_train)

# check the model score !!
clf.score(x_test,y_test)
```

Out[28]:
```
0.639344262295082
```

Random Forest classifier!

In [29]:
```
# Choose RandomForestClassifier (ensemble model)
np.random.seed(seed=42)
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()

# instantiate and fit the model
clf.fit(x_train,y_train)

# check the model score !!
clf.score(x_test,y_test)
```

Out[29]: 0.8524590163934426

Make predictions!

```
In [31]: # Make a prediction
         y_preds = clf.predict(x_test)
         y_preds

Out[31]: array([0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0,
                0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0,
                0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1], dtype=int64)
```

Evaluation Metrics

M1

Accuracy Score

The accurate of our prediction through cross validation score

M2

Confusion Matrix

is a quick way to compare the labels a model predicts and the actual labels it was supposed to predict. In essence, giving you an idea of where the model is getting confused.

M3

ROC Curve

comparison of a model's true positive rate (tpr) versus a models false positive rate (fpr).

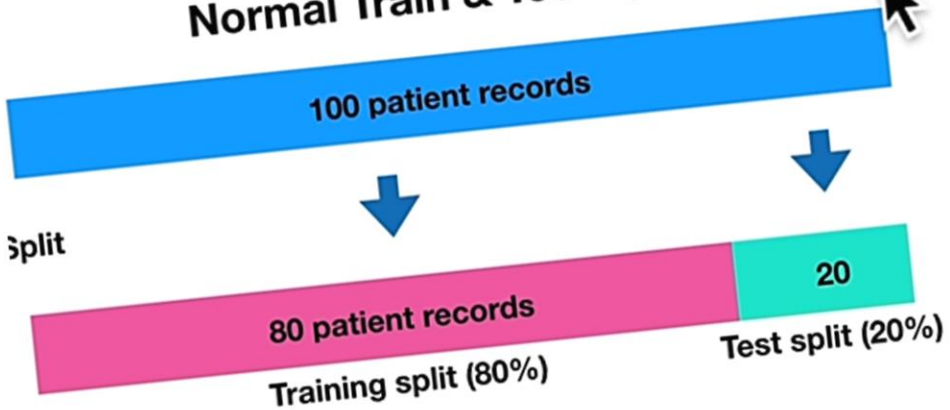
M4

Classification Report

Report contains precision , Recall, F1 Score and other metrics.

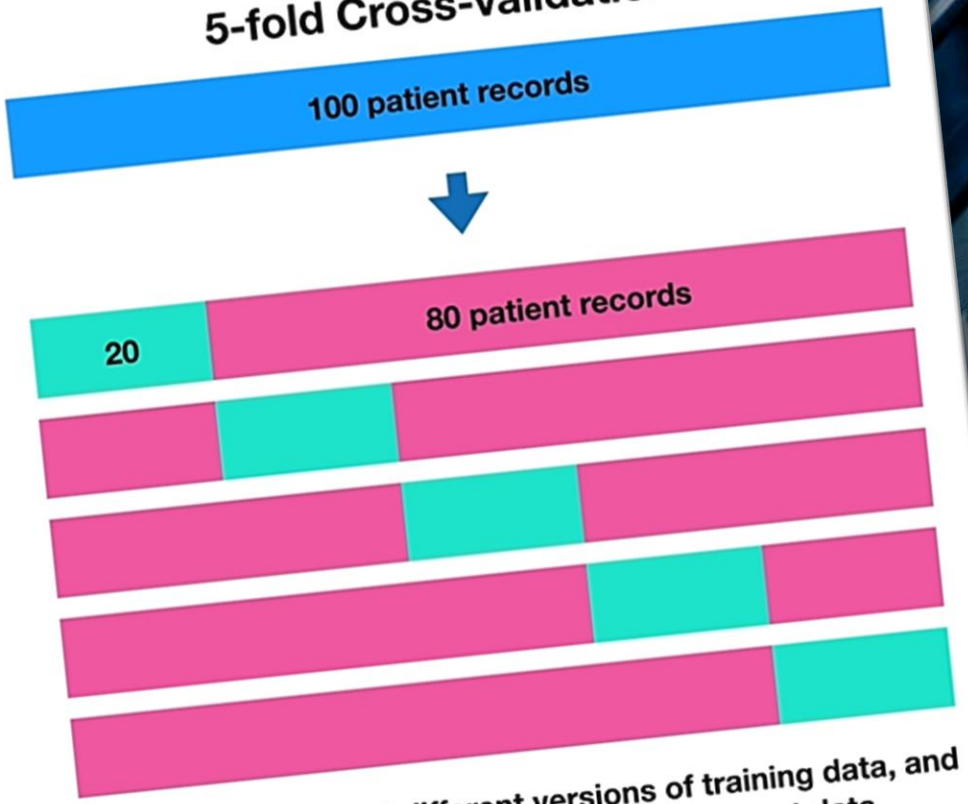
Cross-validation

Normal Train & Test Split



Model is trained on training data, and evaluated on the test data.

5-fold Cross-validation



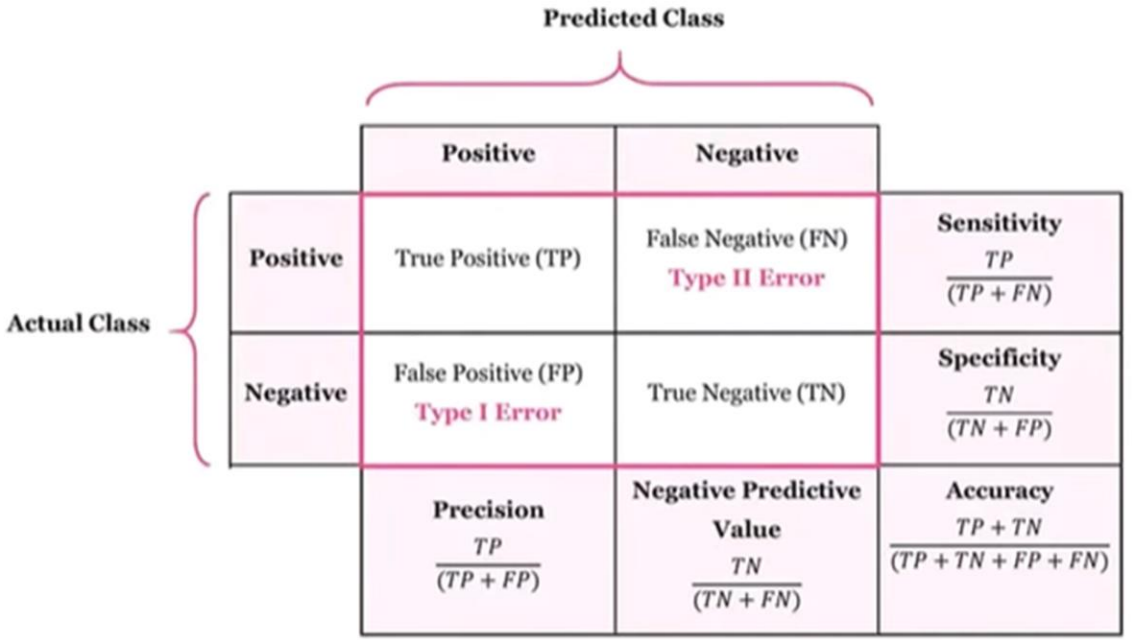
Model is trained on 5 different versions of training data, and evaluated on 5 different versions of the test data.

Cross validation score!

```
In [33]: from sklearn.model_selection import cross_val_score
         cross_val_score = cross_val_score(clf, x, y, cv=5)
         cross_val_score

Out[33]: array([0.83606557, 0.8852459 , 0.83606557, 0.8       , 0.75      ])

In [34]: np.mean(cross_val_score)

Out[34]: 0.8214754098360656
```

Confusion Matrix

- **Confusion Matrix** gives you an idea where the model is getting **confused**.
- It is a quick way to compare model **prediction** with the **actual** data.
- **Accuracy** is the percent of cases classified correctly.
- **Accuracy Rate** = Correct / Total
- **Error Rate** = Wrong / Total



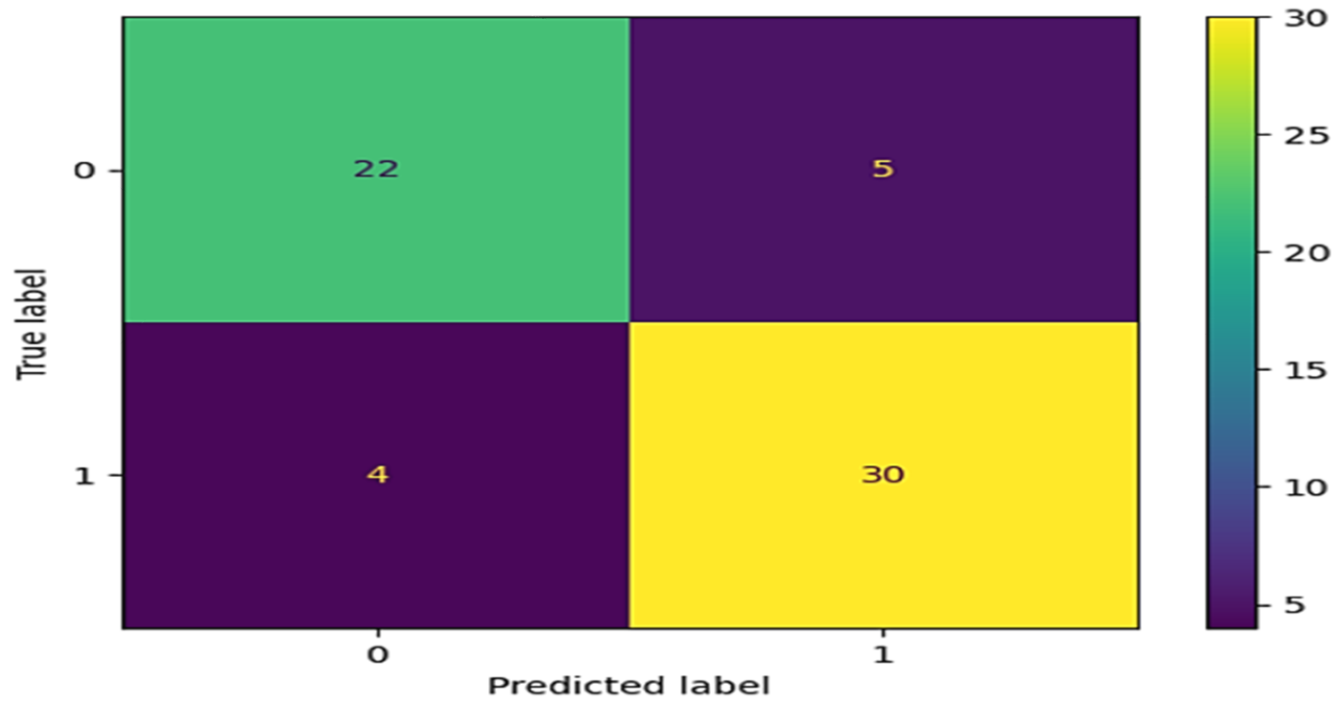
Confusion Matrix | Measures

- **Precision:**
  - The percent of all **1s** that are **actually 1s**
  - **Precision = True Positive / Actual Data**
- **Recall (Sensitivity):**
  - The percent of all **1s** that are correctly **predicted** as **1s**
  - **Recall = True Positive / Predicted Data**
- **F1-Score:**
  - A metric that attempts to measure both precision & recall.
- **Recall & Precision Trade-off**
  - High Recall / Low Precision
  - Low Recall / High Precision

Precision	$\frac{tp}{tp + fp}$
Recall (Sensitivity)	$\frac{tp}{tp + fn}$
F1-score	$\frac{2 * precision * recall}{precision + recall}$

Confusion Matrix!

```
In [42]: from sklearn.metrics import ConfusionMatrixDisplay
         cm = confusion_matrix(y_test, y_preds,labels=clf.classes_)
         disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=clf.classes_)
         disp.plot()
         plt.show()
```



ROC Curve!

RECEIVER OPERATING CHARACTERISTIC CURVE (ROC)

True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

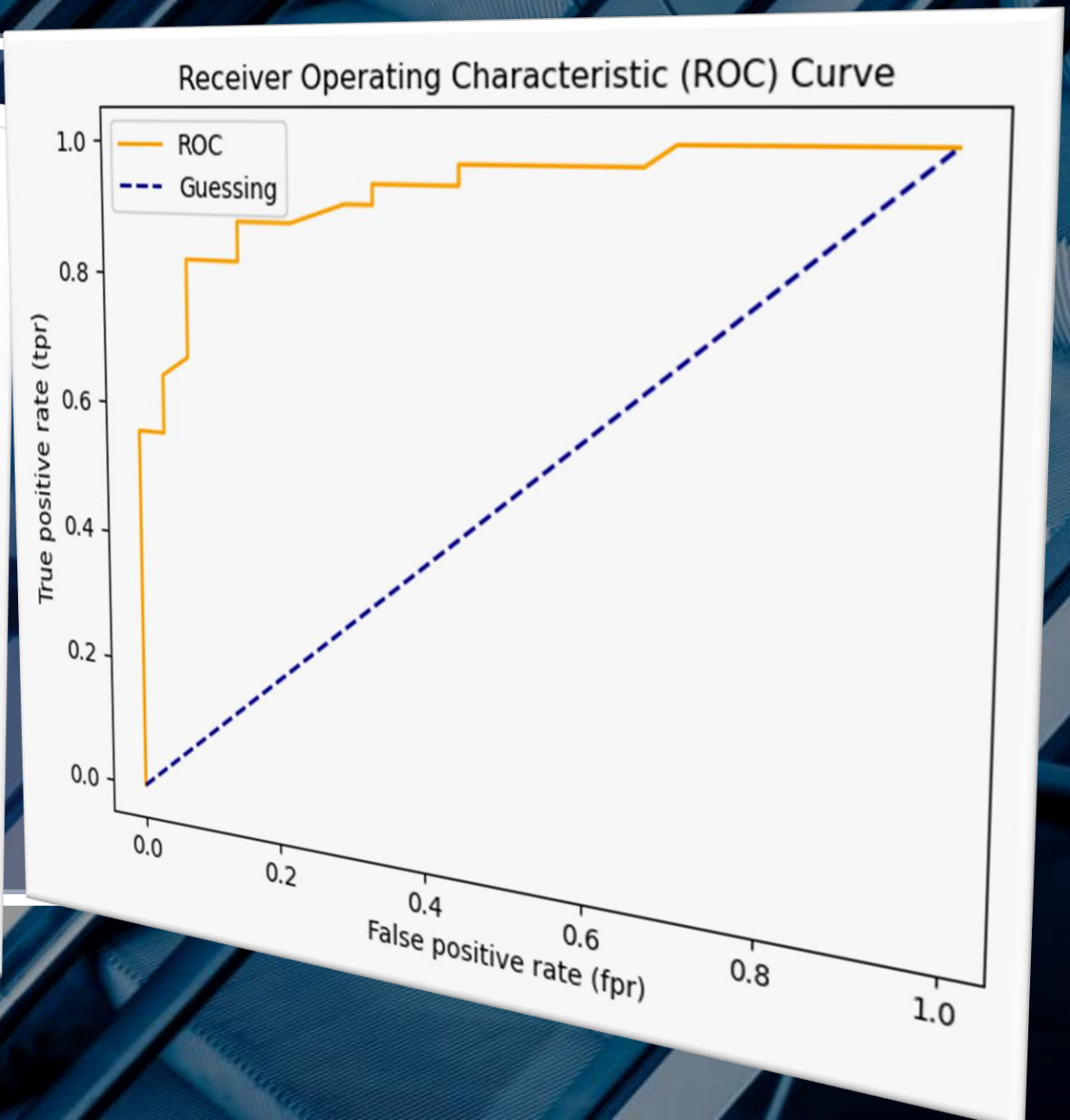
$$\underset{\textbf{Sensitivity}}{TPR} = \frac{TP}{TP + FN}$$

False Positive Rate (FPR) is defined as follows:

$$\underset{\textbf{Specificity}}{FPR} = \frac{FP}{FP + TN}$$

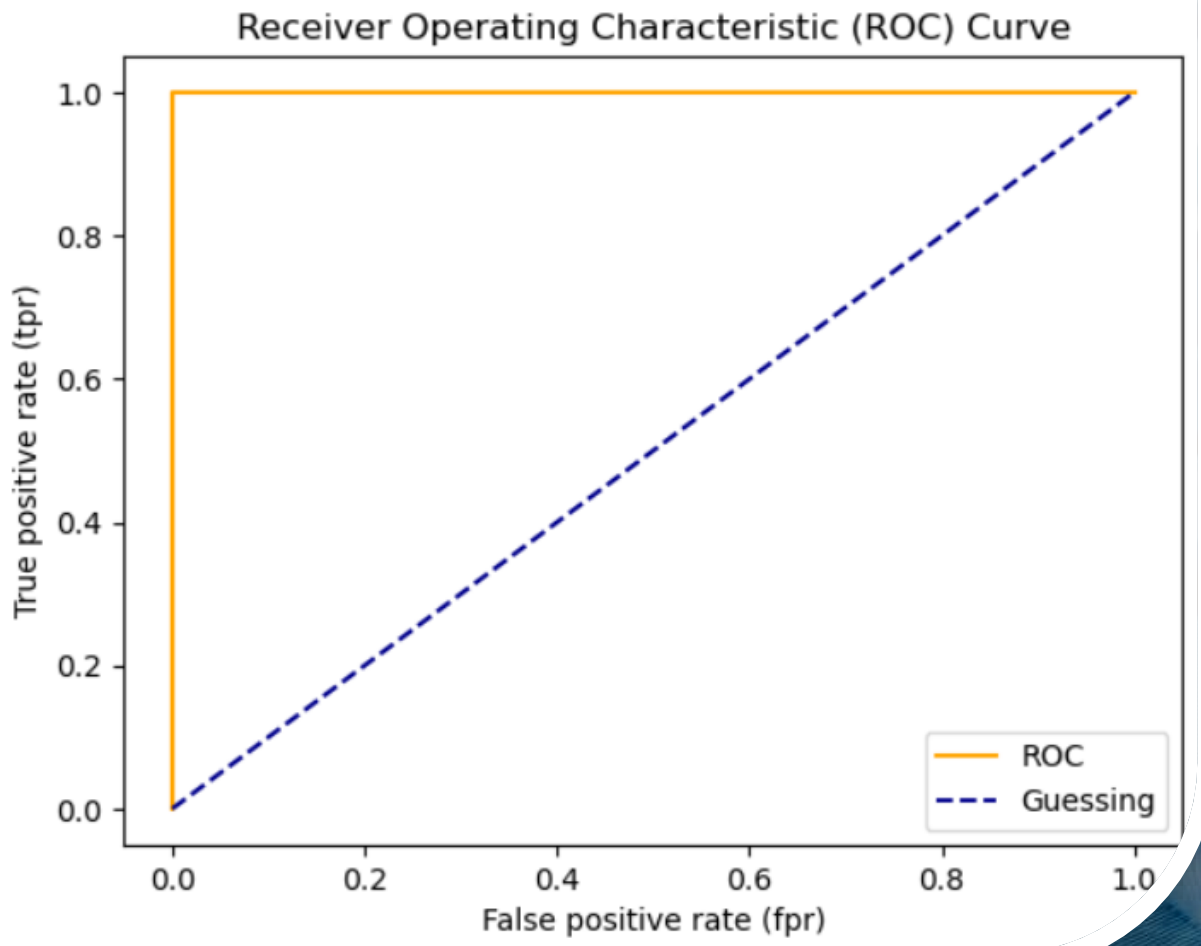
ROC Curve

```python
# Calculate fpr, tpr and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_probs_positive)

# Create a function for plotting ROC curves
import matplotlib.pyplot as plt

def plot_roc_curve(fpr, tpr):
    """
    Plots a ROC curve given the false positive rate (fpr)
    and true positive rate (tpr) of a model.
    """
    # Plot roc curve
    plt.plot(fpr, tpr, color="orange", label="ROC")
    # Plot line with no predictive power (baseline)
    plt.plot([0, 1], [0, 1], color="darkblue", linestyle="--", label="Guessing")

    # Customize the plot
    plt.xlabel("False positive rate (fpr)")
    plt.ylabel("True positive rate (tpr)")
    plt.title("Receiver Operating Characteristic (ROC) Curve")
    plt.legend()
    plt.show()

plot_roc_curve(fpr, tpr)
```



The Perfect ROC Curve be like!

In [47]:
```
# Plot perfect ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_test)
plot_roc_curve(fpr, tpr)
```



Classification report anatomy

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.81      0.90      0.85        29
           1       0.90      0.81      0.85        32

    accuracy                           0.85        61
   macro avg       0.85      0.85      0.85        61
weighted avg       0.86      0.85      0.85        61
```

- **Precision** - Indicates the proportion of positive identifications (model predicted class 1) which were actually correct. A model which produces no false positives has a precision of 1.0.
- **Recall** - Indicates the proportion of actual positives which were correctly classified. A model which produces no false negatives has a recall of 1.0.
- **F1 score** - A combination of precision and recall. A perfect model achieves an F1 score of 1.0.
- **Support** - The number of samples each metric was calculated on.
- **Accuracy** - The accuracy of the model in decimal form. Perfect accuracy is equal to 1.0.
- **Macro avg** - Short for macro average, the average precision, recall and F1 score between classes. Macro avg doesn't class imbalance into effort, so if you do have class imbalances, pay attention to this metric.
- **Weighted avg** - Short for weighted average, the weighted average precision, recall and F1 score between classes. Weighted means each metric is calculated with respect to how many samples there are in each class. This metric will favour the majority class (e.g. will give a high value when one class out performs another due to having more samples).

Improve a model!

```
In [242]: # Improve a model
          # Try different amount of n_estimators
          np.random.seed(42)
          for i in range(10, 100, 10):
              print(f"Trying model with {i} estimators...")
              clf = RandomForestClassifier(n_estimators=i).fit(x_train, y_train)
              print(f"Model accuracy on test set: {clf.score(x_test, y_test) * 100:.2f}%")
              print("")
```

```
Trying model with 10 estimators...
Model accuracy on test set: 85.25%

Trying model with 20 estimators...
Model accuracy on test set: 80.33%

Trying model with 30 estimators...
Model accuracy on test set: 83.61%

Trying model with 40 estimators...
Model accuracy on test set: 80.33%

Trying model with 50 estimators...
Model accuracy on test set: 86.89%

Trying model with 60 estimators...
Model accuracy on test set: 83.61%

Trying model with 70 estimators...
Model accuracy on test set: 83.61%

Trying model with 80 estimators...
Model accuracy on test set: 83.61%

Trying model with 90 estimators...
Model accuracy on test set: 81.97%
```

THANK YOU!

Thanks for giving me this a great opportunity to present myself! And I hope I get your expectations.

Khaled Shaker

khaledgama4@gmail.com

Khaled Shaker