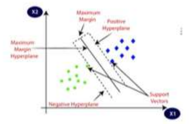
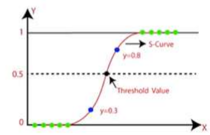
## *K-Fold Cross-Validation*

In K-Fold Cross Validation, we split the dataset into **"K"** number of **folds** (subsets). One chunk of data is used as test data for evaluation & the remaining part of the data is used for training the model. Each time, a different chunk will be used as the test data.

| *K = 5* | Dataset | | | | |
|---|---|---|---|---|---|
| Iteration 1 | Train | Train | Train | Train | Test |
| Iteration 2 | Train | Train | Train | Test | Train |
| Iteration 3 | Train | Train | Test | Train | Train |
| Iteration 4 | Train | Test | Train | Train | Train |
| Iteration 5 | Test | Train | Train | Train | Train |




*Support Vector Machine*




*Logistic Regression*

Helps in model selection

## *K-Fold Cross-Validation*

✓ *Accuracy score for SVM = 84.4 %*

✓ *Accuracy score for Logistic Regression = 88 %*

| | | | | | |
|---|---|---|---|---|---|
| Iteration 1 | Train | Train | Train | Train | Test |
| Iteration 2 | Train | Train | Train | Test | Train |
| Iteration 3 | Train | Train | Test | Train | Train |
| Iteration 4 | Train | Test | Train | Train | Train |
| Iteration 5 | Test | Train | Train | Train | Train |

### *Advantages of using K-Fold Cross-validation:*

- Better alternative for train-test split when the dataset is small
- Better for multiclass classification problems

Instead of using K-fold from sklearn to do this
Use cross_val_score it is much simplier to determine which model to use

```
from sklearn.model_selection import cross_val_score
```

```
cv_score_svc = cross_val_score(SVC(kernel = "linear"),X , Y ,cv = 5)
mean_accuracy_svc = sum(cv_score_svc)/len(cv_score_svc)
mean_accuracy_svc = mean_accuracy_svc*100
mean_accuracy_svc = round(mean_accuracy_svc,2)
print(mean_accuracy_svc)
```

Use this function

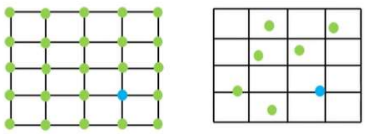
```
models = [LogisticRegression(max_iter = 1000), SVC( kernel = 'linear'),
KNeighborsClassifier(),RandomForestClassifier()]
```

```
def compare_models_cross_validation():
  for model in models:
    cv_score = cross_val_score(model, X,Y, cv= 5)
    mean_accuracy = sum(cv_score)/len(cv_score)
    mean_accuracy = mean_accuracy*100
    mean_accuracy = round(mean_accuracy,2)
    print("The accuracy of the ",model, "=", mean_accuracy)
    print("---------------------------------------------")
```

## *Hyperparameter Tuning*

**Hyperparameter Tuning** refers to the process of choosing the optimum set of hyperparameters for a Machine Learning model. This process is also called **Hyperparameter Optimization.**

### *Hyperparameter Tuning Types:*



GridSearchCV RandomizedSeaechCV

**Support Vector Classifier:**
C: [1,5,10]
kernel: ('linear', 'poly', 'rbf', 'sigmoid')

It used to select the best hyperparameters in the machine learning models as in Support Vector Classifier

Random Search CV

```
[18] X = np.asarray(X)
     Y = np.asarray(Y)
```

```
[19] model = SVC()
```

```
[20] hyperparameters = {
         'kernel' : ['linear','poly','sigmoid'],
         'C': [1,5,10,20]
     }
```

```
[21] classifier = RandomizedSearchCV(model, hyperparameters, cv = 5)
```

```
classifier.fit(X,Y)
```

```
[ ] classifier.cv_results_
```

```
[ ] classifier.best_params_
```

Used Only in binary calssification (accuracy score and confusion matrix)

## *Accuracy Score*

In Classification, **Accuracy Score** is the ratio of **number of correct predictions** to the **total number of input data points**.

$$\text{Accuracy Score} = \frac{\text{Number of correct predictions}}{\text{Total Number of data points}} \times 100 \ \%$$

Number of correct predictions = 128

Total Number of data points = 150

Accuracy Score = 85.3 %

from sklearn.metrics import accuracy_score

On training data

## *Limitation of Accuracy Score*

**Accuracy Score** is not reliable when the dataset has an uneven distribution of classes

Number of dog images = 800

Number of cat images = 200

Number of images predicted as dog = 1000

Number of images predicted as cat = 0

Number of correct predictions = 800

Total Number of data points = 1000

$$\text{Accuracy Score} = \frac{800}{1000} \times 100\ \%$$

Accuracy Score = 80 %

On testing data

## *Limitation of Accuracy Score*

**Accuracy Score** is not reliable when the dataset has an uneven distribution of classes

Test data: Number of dog images = 200

Number of cat images = 200

Number of images predicted as dog = 400

Number of images predicted as cat = 0

Number of correct predictions = 200

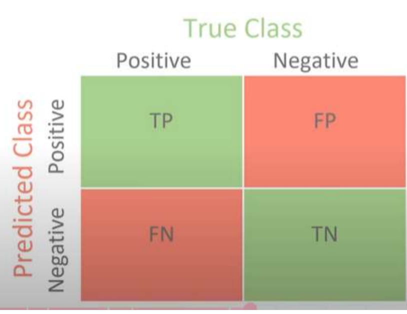
Total Number of data points = 400

$$\text{Accuracy Score} = \frac{200}{400} \times 100\ \%$$

Accuracy Score = 50 %

## *Confusion Matrix*

Confusion Matrix is a matrix used for evaluating the performance of a Classification Model. It gives more information than the accuracy score.




TP + TN = Correct Predictions

FP + FN = Wrong Predictions

sklearn.metrics.confusion_matrix

Confusion Matrix

```
[17] from sklearn.metrics import confusion_matrix
```

```
[18] cf_matrix = confusion_matrix(X_train_prediction, Y_train)
     print(cf_matrix)
```

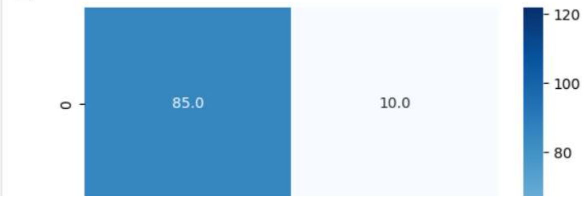
```
[[ 85  10]
 [ 25 122]]
```

Heatmap for Confusion Matrix

```
[19] import seaborn as sns
```

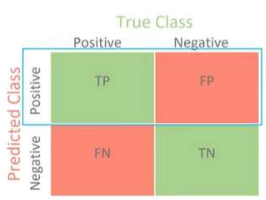
```
sns.heatmap(cf_matrix, annot = True, cmap = 'Blues', fmt=".1f" )
```

```
<Axes: >
```




## Precision

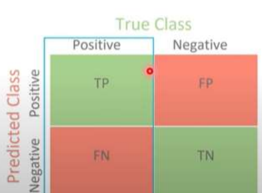
$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \textbf{False Positive}} \Rightarrow \text{Precision} = \frac{\text{True Positive}}{\text{Total Predicted Positive}}$$




**Precision** is the ratio of number of **True Positive** to the **total number of Predicted Positive**. It measures, out of the total predicted positive, how many are actually positive.

## Recall

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \Rightarrow \text{Recall} = \frac{\text{True Positive}}{\text{Total Actual Positive}}$$



## *F1 Score*

**F1 Score** is an important evaluation metric for binary classification that combines Precision & Recall. F1 Score is the **harmonic mean** of Precision & Recall.

This is a very useful metric when a dataset has imbalanced classes.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

## *Precision, Recall & F1 Score*

**Example:**

**Predicted**

| Actual | | Positive | Negative |
|---|---|---|---|
| | Positive | TP = 50 | FN = 10 |
| | Negative | FP = 5 | TN = 20 |

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} = \frac{50}{50 + 5}$$

**Precision = 0.91**

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} = \frac{50}{50 + 10}$$

**Recall = 0.83**

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{0.91 \times 0.83}{0.91 + 0.83}$$

**F1 Score = 0.87**

SUBSCRIBE

Creating a function for precision, recall and f1_score

```
[38] def precision_recall_f1score(true_labels, pred_labels):
       precision_value = precision_score(true_labels, pred_labels)
       recall_value = recall_score(true_labels, pred_labels)
       f1_score_value = f1_score(true_labels, pred_labels)

       print("precision =" , precision_value)
       print("recall =" , recall_value)
       print("f1_score =" , f1_score_value)
```

```
[39] precision_recall_f1score(Y_train,X_train_prediction)
```

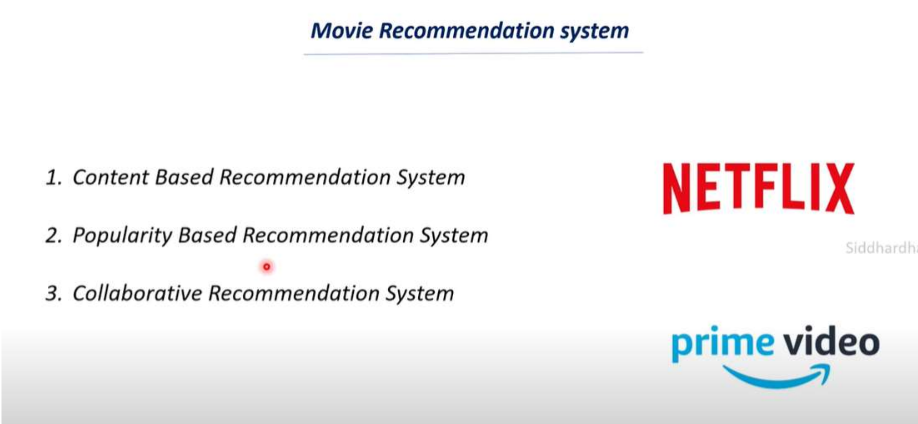
```
precision = 0.8299319727891157
recall = 0.9242424242424242
f1_score = 0.8745519713261649
```

```
precision_recall_f1score(X_test_prediction, Y_test)
```

```
precision = 0.8181818181818182
recall = 0.8181818181818182
f1_score = 0.8181818181818182
```

**Otherwise can find individually similar how you find accuracy score**

**In Data Visualization use countplot for categorical features such as age**
**And distplot for numerical columns like age, etc**




1. **Content based = gives recommendation on the movies we watched**
2. **Popularity based = gives recommendation on movies which are popular**
3. **Collaborative = groups people who have similar taste in movies and show**

**For movie recommendation system we don't use ML model we use cosine similarity to find similarity in vectors**