# Ensembles

Combine multiple machine learning model to create a power full models.

Tree-based models :  XGBoost, LGBM and CatBoost(**permutation sampling for categorical value)**

Permutation sampling for categorical values is a method for creating synthetic samples from the original data set to increase the diversity of the data and prevent.

In permutation sampling for categorical values, CatBoost randomly permutes the values of the categorical features within a specified number of bins. This allows the model to learn the **relationship between the categorical feature and the target variable from multiple different perspectives**, and can lead to more robust and accurate predictions.

The use of permutation sampling can help improve the performance of the model on categorical features, especially when the values of the categorical features are not ordinal.

**Ensemble approaches** : Averaging and Stacking( learn from mistake of previous model mistake)

**Averaging**: Combining predictions by taking average, example: **Voting Classifier**, Averaged Perceptron, Soft Voting, Weighted Average Ensemble, Bagging Classifier

**Stacking**: Training a new model using previous models predictions, example: **StackNet**, Super Learner, Blending, Stacked Generalization, Stacked Ensemble.

It is particularly **useful for datasets with large number of features and categorical variables.**

## **Ensemble methods are techniques**:

**Bagging stands** for Bootstrap Aggregating: Training multiple models independently on random **subsets with replacement and parrallel training, example: Random Forest**, Extra Trees, Bagged Decision Trees, **Random Forest Classifier, Bagging Classifier.**

Bagging Classifier is a special case where it combines the predictions of multiple classifiers by taking the majority vote, which is a form of averaging and it trains multiple models independently on random subsets of the data, which is the definition of bagging.

**Boosting**: Training multiple models **sequentially** to correct previous errors, example: **XGBoost, AdaBoost, Gradient Boosting Machine (GBM), LightGBM, Catboost**

In summary, ensemble methods such as **Voting Classifier,** StackNet, Random Forest and XGBoost are popular implementations of the ensemble approaches of Averaging, Stacking, Bagging, and Boosting respectively. These methods are implemented in various libraries and can be used to improve the performance of a single model by **combining the predictions of multiple models**.

## Keys diff between three models:

Speed and performance : XGBoost, LightGBM faster and catboost is **slower**(designed for categorical variables)

Memory efficients : Catboost is more memory efficient and can **large** datasets that may not fit in the model.

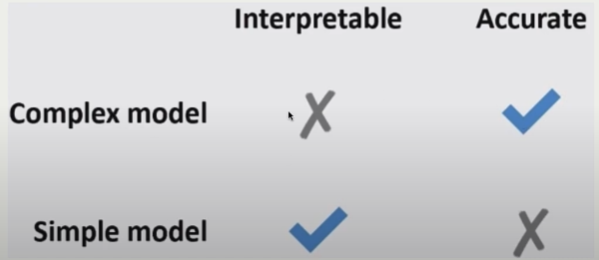
# Benefits of Tree based models

* Interpretable
* Capture non linear relationships
* No need to scale the data.
* Handle missing values and categorical features.
* Single decision can overfit.

## What is Interpretable model?

* Turst : Human can understand the model
* Debugging: Easy debugging
* Regulatory compliance : Finance and health care regulation check
* Improved understanding : can easily find underneath pattern of data

1. **LR is simple but interpretable.**
2. Highly used in banking, health, criminal justice.
3. It can be used domain expert to match the result.
4. We can consider model interpretablity as features importances but there are many more.



# Transformer and pipeline

numeric\_transformer = StandardScaler()

ordinal\_transformer = OrdinalEncoder(categories=[education\_levels], dtype=int)

binary\_transformer = make\_pipeline(

SimpleImputer(strategy="constant", fill\_value="missing"),

OneHotEncoder(drop="if\_binary", dtype=int),

)

categorical\_transformer = make\_pipeline(

SimpleImputer(strategy="constant", fill\_value="missing"),

OneHotEncoder(handle\_unknown="ignore", sparse=False),

)

preprocessor = make\_column\_transformer(

(numeric\_transformer, numeric\_features),

(ordinal\_transformer, ordinal\_features),

(binary\_transformer, binary\_features),

(categorical\_transformer, categorical\_features),

#(“pass\_through”, example)

("drop", drop\_features),

)

# Decision tree classifier

It works by creating a tree-like model of decisions based on the features of the data.

Prone to overfitting

# Random forest(bagging technique)

Ensembles method combine multiple decision tree.

* n\_estimators : how many decision tree
* fit : inject randomnes in the model
* Using bootstraping with replacement for creating different dataset to overcome overfitting
* Take majority vote of individual DT to make final prediction.
* predict : voting(classification) and averaging(regression)
* best performing without heavy tunning and don’t need scaling

## Inject **randomness** in the classifier construction

To ensure that the trees in the random forest are different we inject randomness in **two ways**:

* **Data**: Build each tree on a **bootstrap sample (**i.e., a sample drawn with replacement from the training set)
* **Features**: At each node, select a **random subset of features** (controlled by max\_features in scikit-learn) and look for the best possible test involving one of these features

## Random forest vs Gradient boosting

#### RF

1. Random forest works on bagging technique( bootstrap aggregating)
2. RF uses different datasets and then avg their final predictions
3. RF uses decesion tree
4. All tree are independent
5. Parrallel training of all tree.

#### GB

* GB works on boosting technique.
* Tree are serial connected.
* No randomization, use prepruning(early stopping by using the learning rate)
* GB works tree is trained to correct the error of the previous tree.
* GB tree is more complex then the DT, it try **to reduce the error or residual of the tree**.
* GB use **iterative** process to reduce the error and iterate until desired result is not achieved
* **Main parameter n\_estimater(higher, complex tree) and learning\_rate(control mistake of previous tree, higher value greater penalty)**

# Cofficients of models

transformed\_example = preprocessor.transform(test\_example)

pd.DataFrame(data=transformed\_example.flatten(), index=feature\_names)

# make\_num\_tree\_plot

number of trees and fundamental tradeoff

**ExtraTreesClassifier**

# Decision tree

Max\_depth, min\_samples\_split, min\_samples\_leaf, max\_leaf\_nodes

* Decide best split on each node
* Try to minimize impurity
* **Common way to minimize impurity(this also help to find the feature importnace) : gini index,cross entropy, information gain, mse**
* Pure node : not impurity

Pros:

* Little data preparation.
* Can handle categorical, numeric,
* Not required data scaling or normalization
* Handle **high dimensional data**
* Stop growing when data is pure

Cons

* Overfitting when tree become deep and complex
* Unstable on small data changes
* Computationally expensive

# Parameter(learned from data) vs Hyper parameter(help to learn)

**Parameters** : internal config, values learned from data like intercept, slope

Hyperparameter**(control how much to learn)** : external config, value not depends upon data. It controls the process of the learning like learning rate and regularization param. Set before training.

Parameters:

* Weights and biases in a neural network
* Slope and intercept in a linear regression model
* Mean and standard deviation in a Gaussian distribution

Hyperparameters:

* Learning rate in a neural network
* Regularization coefficient in a linear regression model
* Number of clusters in a k-means clustering algorithm
* Depth of a decision tree in a random forest model

# Gradient boosted trees

* No randomization
* Combine weak leaners to create a strong learner
* Each trees tries correct amistakes of previous

## XGBoost

* Support missing values
* Gpu training
* Network **parrallel training**
* Support **sparese matrix.**

## LightGBM

* Small and faster model

## Catboost

* Better compared to both model but slower.

## Important param

n\_estimators : number of boosting rounds

learning\_rates : how strongly each tree should be corrected higher is number complex is model.

Scale\_pos\_weight: balance positive and negatives weights.

# Bias(completely unable to understand) and Variance(understand not perfect always)

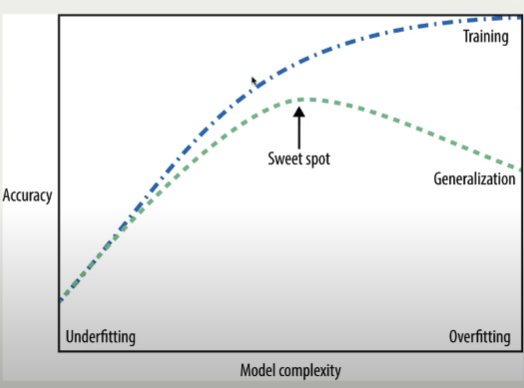
**Bias** : learning same wrong thing again again; inability to accurately capture the underlying patterns of the data.

Predict height of person based on weight : a model with higher bias might unable understand relationship

**High bias : 1. High training error; 2. Training error close to test error**

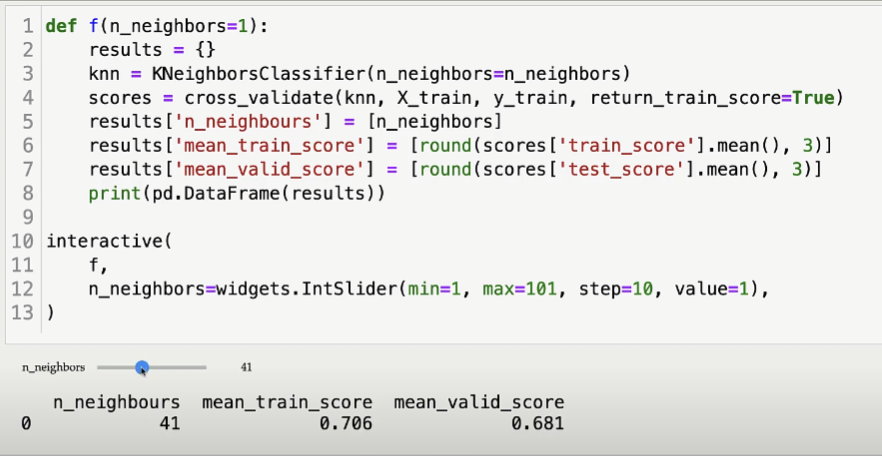
**Variance** : learning random things not real siganl; model sensitivity to specific detials of the data.

**High variance : 1. Low training error 2. Training error much lower thn test error**



# KNN

Interactive for call with slider value good example to remember:



# SVM vs KNN

Hperparameter : c, gamma 🡪 control fundamental tradeoff for Try to find best decision boundary

SVM(hyperplane; only r, c & eager learning algorithm) and KNN(set of nearest neogbhours ; only in c ,lazy learning algorithm) both are non linear

KNN : sensitive to the feature with large scale of the range, they dominate

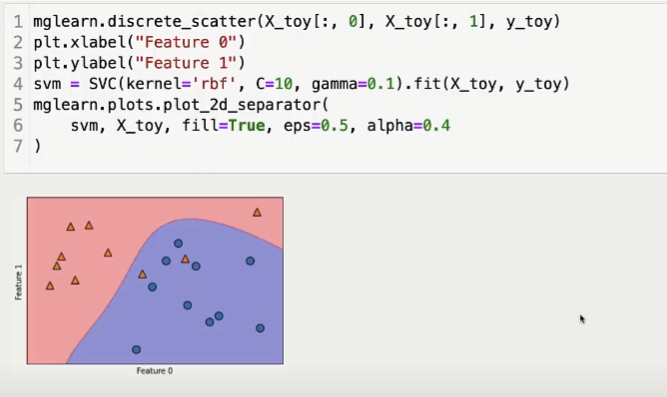
SVM : it uses kernel function to project the data in the higher dimensional

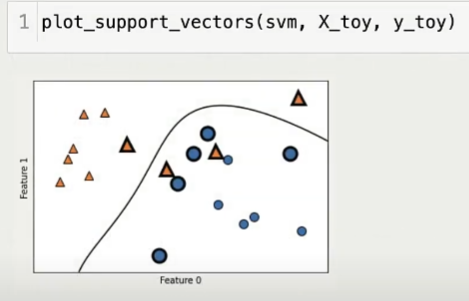
For distance kNN uses Eculidian distance whereas SVM uses Kernal

## Kernal type(different type of boundry):

* Linear; Polynomial; Radial bias fucntion(RBF); Laplace; Sigmoid

good to remember this diagram code:

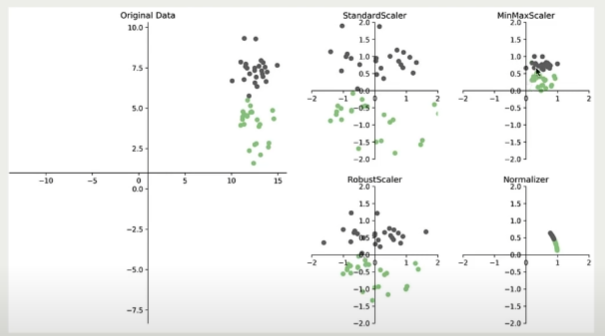




Larger gamma or c : more complex model; smaller gamma or c : less complex model

How do we find best of both, by using GridSearchCV and RandomSearchCV

# Scaler fucntion diagram in ML:



# OneHotEncoder

OneHotEncoder(handle\_unknown="ignore", sparse=False)

Handle\_unknown used to handle categories that are present in the test data but not in the training data when encoding categorical features.

OneHotEncoder(drop="if\_binary", dtype=int)

It will drop third category if present in the binary data and will consider only first two.

# Ordinal encoder

If there is ordinality in the data like child schools classes

Preschool, LKG, UKG, 5th, 10th, degree

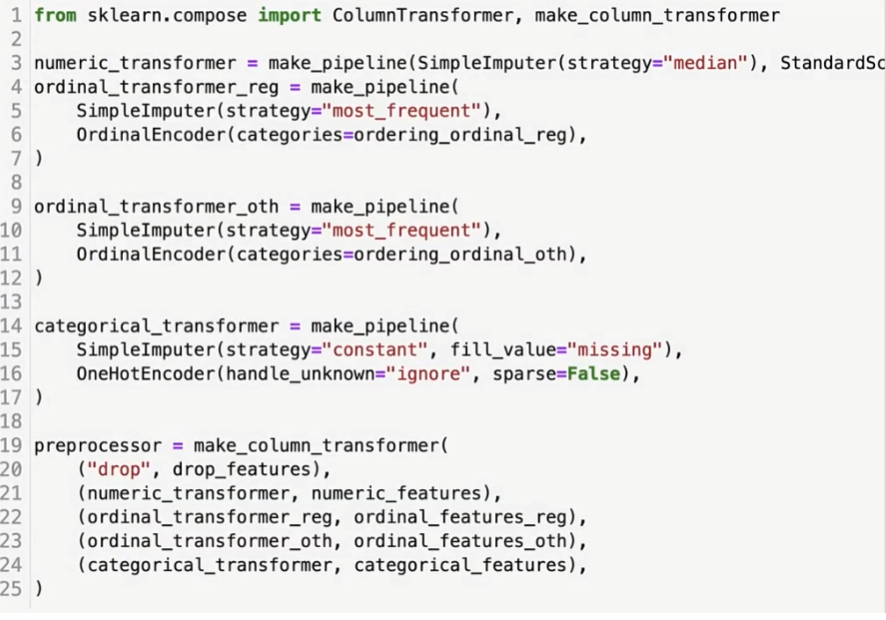
Example like language english, hindi, french, urdu

Output : 0,1,2,3

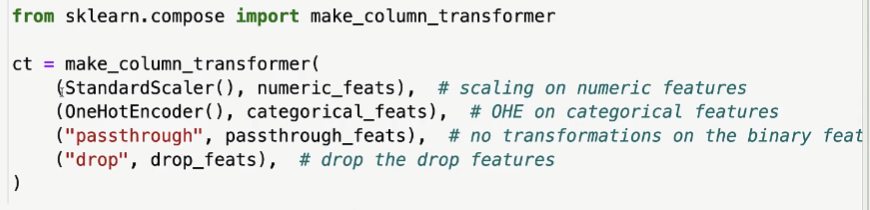
We know english is not closure to hindi and french but as per ordinal encoder, all these three are close. Which is invalid.

We need to pass same ordinal order for same type ordinal features.

[ordering] \* len(ordinal\_features\_reg)



# Pass through and Drop feature:



Ct.fit\_transformer() :: not mandatory

Ct.named\_transformer\_[“onhotencoder”].get\_features\_names().tolist()

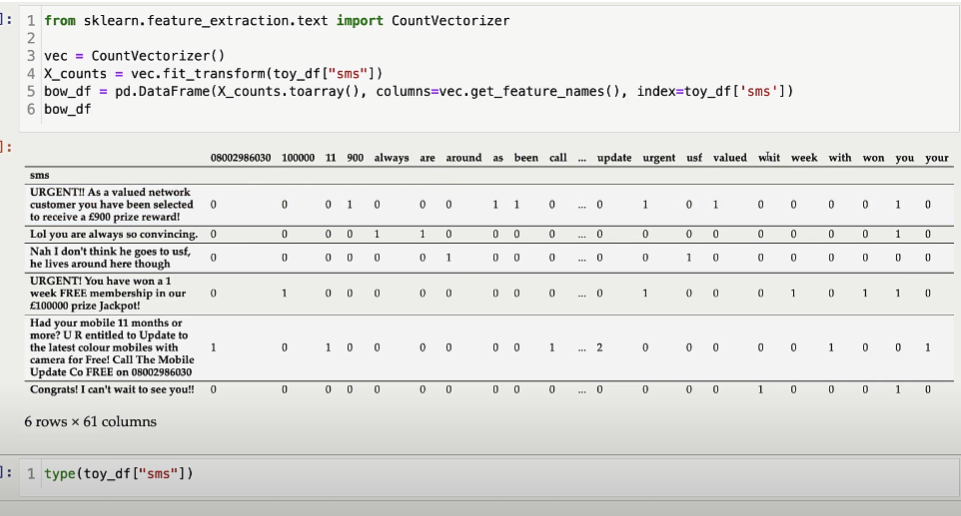
Ct.named\_transformer

# BOW using CountVectorizer



To extract bow features using scikit-learn Countvectorizer

## Good eg to remember



If we check more deeply output we can see that it is not 1 or 0 it is actual count of the word check secnod last row. For update value is 2

X\_counts 🡪 compressed sparse row format

We can explore the X\_counts variable like

X\_counts.nnz : non zero elements

## Hyperparameter:

Binary=true/false, max\_features , max\_df, min\_df, ngram\_range

Binary = true will convert 2 into 1 now it follow binary rule.

Max-features : control vocab size

# Linear model

Linear model learn coefficients associated with them which tells us the importance of the features.

They make predictions using linear function of the input features.

* Linear regression
* Logistic regression
* Linear SVM

## Ridge(HP C)

To overcome ridge **overfitting issue penalty** is applied to loss function. How much penalty needed to apply is controlled by hyperparamter alpha.

It determines amount shrinkage on the cofficients.

Coef with 0 value signify not important featrues, meaning less impact.

Large alpha 🡪 large penalization 🡪 higher shrinkage towards 0.

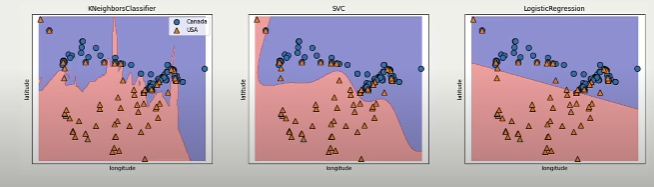
Small alpha 🡪 small penalty 🡪 less shrinkage 🡪 overfitting model

To find better trade off we can implement cross validation hyperparameterization.

## Logistic regression(HP C):

LG is simple but interpretable.

Highly used in banking, health, criminal justice.



Why do we need logistic reg if it is so restrictive 🡪 low dimensional sapce

In multiple element with multi dimenesional space it beocme powerful.

C hyperparameter control fundamental tradeoff

LG Interpretaion is similar to C in SVM RBF opposite of Logistic regresssion

LG == SVM RBF != Ridge

Smaller c 🡪 might lead to underfitting

Large c 🡪 overfitting

If model overfitting in validation dataset with cross validation then try to collect more data.

# Recall

Among all positive how many all positive all actually indentified.

Recall = TP / (TP + FN) = TP / (all positive)

# Precision

Among all indentified positive how many are actually positive.

Precision = TP / (TP + FP)

# F1 score

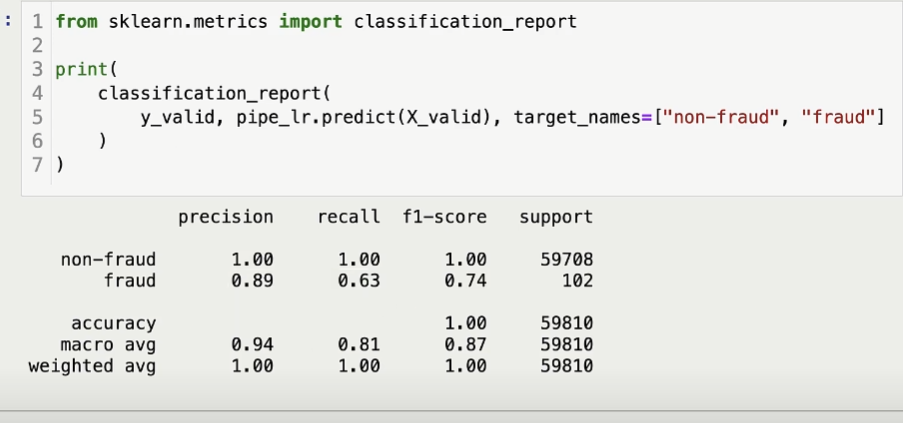
Can be used for the hyperparametrization as it is harmonic mean of both prcision and recall.

F1 = 2 \* (precision \* recall) / (precision + recall)

From sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score. Classification\_report.

## Class important to remember

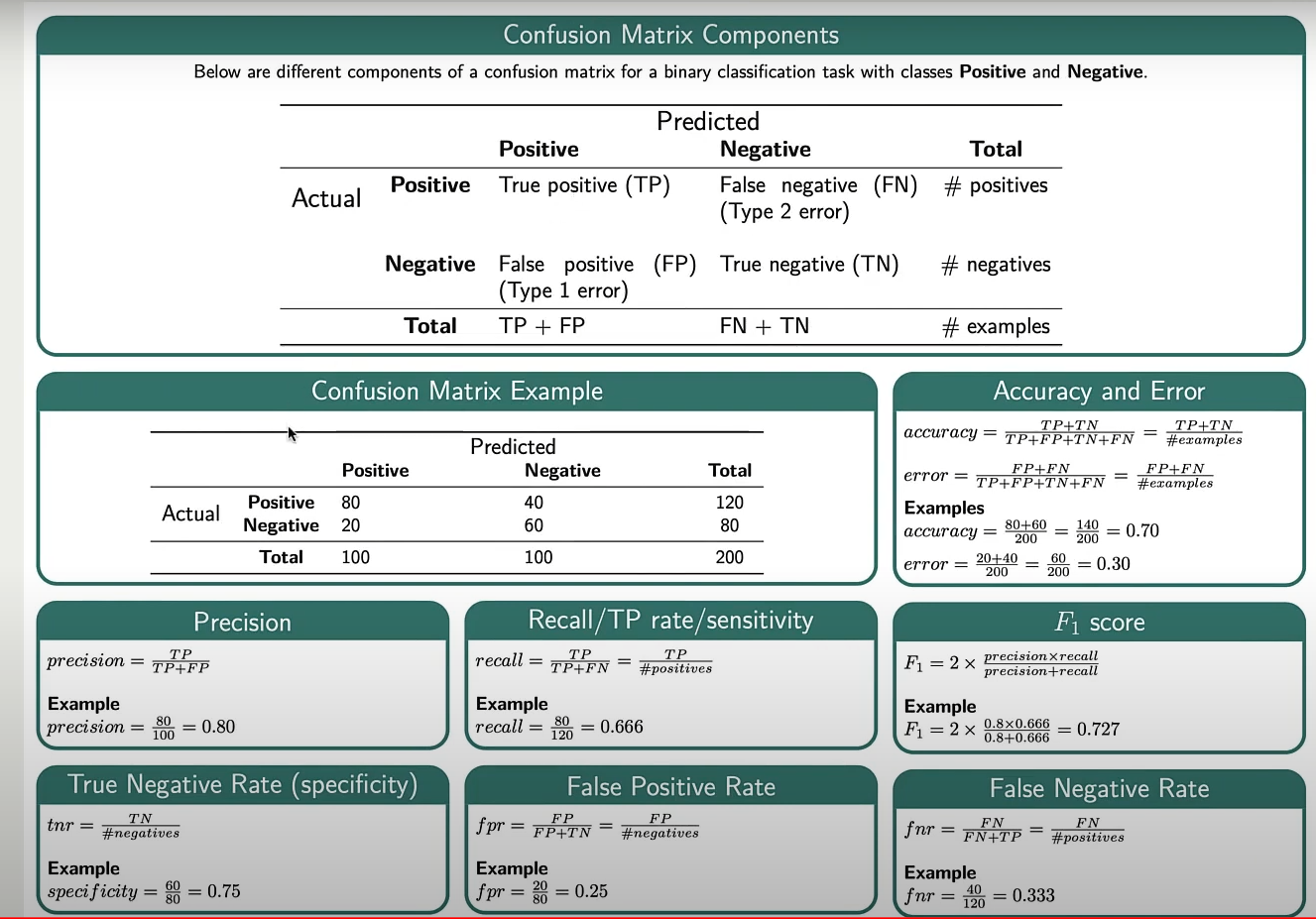
1. Classification\_report
2. Plot\_confusion\_matrix



Acuuracy is misleading when there class imblance.

Macro avg give equal weight to each class , equal weight to fraud and non fraud

Weighted avg gives eqyal weight to each example



## Eg of positive & negative:

positive class is important in classification tasks, with short names:

* Precision is beneficial when false positive is costilier for the model final result.
* Precision is important to minimize the number of false positive predictions
* High precision would be desirable

1. Employee retention (positive class)
2. Customer conversion (positive class)
3. **Disease risk prediction** (positive class)
4. Candidate selection (positive class)
5. Real estate sales prediction (positive class)
6. Investment performance prediction (positive class)
7. Product demand prediction (positive class)
8. Social media engagement prediction (positive class)
9. Project success prediction (positive class)
10. Online ad engagement prediction (positive class)

negative class is important in classification tasks, with short names:

* Recall is beneficial when the hight false negative prediction is costilier.
* Recall is important to minimize the number of false negative predictions.
* High recall would be desirable.

1. **Credit card fraud detection** (negative class)
2. Medical test accuracy (negative class)
3. **Email spam filtering** (negative class)
4. Aircraft maintenance prioritization (negative class)
5. Online review authenticity (negative class)
6. Security threat assessment (negative class)
7. Food safety (negative class)
8. Power outage root cause analysis (negative class)
9. News article authenticity (negative class)
10. Machine failure prediction (negative class)

Train\_test\_split 🡪 stratified split 🡪 if 10% total negative then each split will get that 10% 🡪 default

Cross\_validation 🡪 Stratified K- Fold cross validation 🡪 same as above

Is stratified good 🡪 not random sampling with this, so this is not good as theortical.

## Pandas-profiling

conda install -c conda-forge pandas-profiling

# Features importance:

There will be not sign with the non-linear features like minus(-).

## Linear SVM, KNN, and RBF SVM

feature importance is measured using **permutation importance, recursive feature elimination, or sensitivity analysis**

## Logistic regression( both positive and negative)

By cofficients of each feature. The larger the coefficient, the more impact the feature has on the prediction made by the model.

## Naive Bayes

By **Probability** estimates for each feature can be used to calculate the importance of the features.

## Linear models

By coefficients of the feature. The larger the **coefficient**, the more impact the feature has on the prediction made by the model.

## Decision trees

By calculating mean decrease impurity, mean decrease accuracy, mean decrease in Gini, or feature importance.

**Like for DT, pipe\_dt.named\_steps[“dc”].features\_importance\_ : FI comes from gini index.**

## Random forests

By calculating the mean decrease in impurity or the mean decrease in accuracy.

## Boosted trees

By calculating the feature importance **scores** provided by the model.

## Neural networks

sensitivity analysis or permutation importance.

# Interpretability in terms of feature importances

There is no perfect way to calculate the interpretability.

* sklearn's feature\_importances\_ and permutation\_importance
* eli5 (stands for "explain like I'm 5")
* SHAP

## EL5

* Provide tools to interprete and explain machine models.
* **It can be used for non sckit learn model to interprete the model(CatBoost, LGBM, XGB).**
* Pip install el5
* For sklean model use, Eli5.explian\_weights() : return explaination object.

**feature\_importances = eli5.explain\_weights(model, feature\_names=feature\_names)**

for weight in **feature\_importances.targets[0].feature\_weights.pos**:

print(f"Feature: {weight.feature} Importance: {weight.weight}")

* For non sklean model use, Eli5. explain\_prediction() : return explaination object.

# Make a prediction with the model

prediction = model.predict(X\_test[0])

**# Explain the prediction**

**explanation = eli5.explain\_prediction(model, X\_test[0], feature\_names=feature\_names)**

# Access the feature importances

for feature in explanation.**features**:

print(f"Feature: {feature.feature} Importance: {feature.weight}")

# Access the raw prediction values and the final prediction value

print(f"Raw predictions: {explanation.raw\_predictions}")

print(f"Final prediction: {explanation.predicted\_class}")

## Pearson correlation

* It just check linear association between the variables.
* Better to the multi colinearity using VIF(Variance inflation factors)

## Shap

This is not good for all stituations model with large datset with many correlated features insight is not possible.

* Why model is doing this
* Can explain the **prediction** of an example by computing the contribution of each feature to the prediction.
* When we want to explain the models to non technical person
* When want to understand which feature is more important.
* It is easy to understand.

It also provides explainer for different kinds of models.

* TreeExplainer (supports XGBoost, CatBoost, LightGBM)
* DeepExplainer (supports deep-learning models)
* KernelExplainer (supports kernel-based models)
* GradientExplainer (supports Keras and Tensorflow models)

Steps

1. create shap explainer object-

explainer = shap.TreeExplainer(fitted\_model)

shap\_values = explainer.shap\_values(X\_test)

1. plot the shap graph

shap.force\_plot(

lgbm\_explainer.expected\_value[1],

# expected value for class 1.

test\_lgbm\_shap\_values[1][ex\_l50k\_index, :],

# SHAP values associated with the example we want to explain

X\_test\_enc.iloc[ex\_l50k\_index, :],

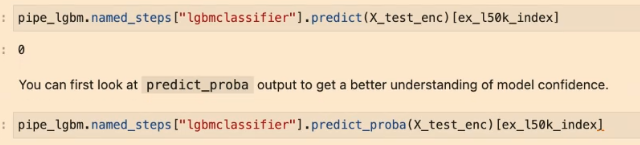
# Feature vector of the example

matplotlib=True,

)

1. Global feature importance using SHAP
2. shap values dependency plot
3. class wise shap vlaues
4. avg impact of the output values

predict on pipeline:



## Guideline

Here are some guidelines and important points to remember when you work on a prediction problem where you also want to understand which features are influencing the predictions.

* Examine multicoliniarity in your dataset using methods such as VIF.
* If you observe high correlations in your dataset, either get rid of redundant features or be mindful of these correlations during interpretation.
* Be mindful that feature relevance is not clearly defined. Adding/removing features can change feature importance/unimportance. Also, feature importances do not give us causal relationships. See this optional section from Lecture 4.
* Most of the models we use in ML are regularized models.

1. With L2 regularization, the feature importances are **distributed evenly among correlated features**.
2. With L1 regularization**, one of the correlated features gets a high importance** and the other gets a lower importance.

* Don't be overconfident. Always take feature importance values with a grain of salt.

# K-Mean Clustering(n\_clusters):

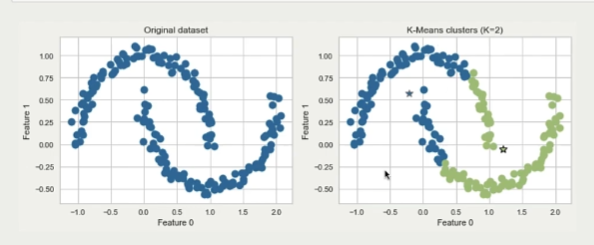
* Unsupervised algorithm to find cluster in dataset.
* Assign predefind number of cluster and locate them in the cluster
* Calculate the distance and draw median line
* Calculate again mean(center of gravity) of both cluster and draw median line .
* Repeat until no new cluster can be created.
* Sklearn provides scaleable variant called **MiniBatchKmenas** which **handle large datasets**

## Limitation

Relies on random initialization

Sil and elbow is not easily interpretable

It fails to indentify the cluster with complex shape.



## **Goodness of fit**

### **WCSS**

stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster.

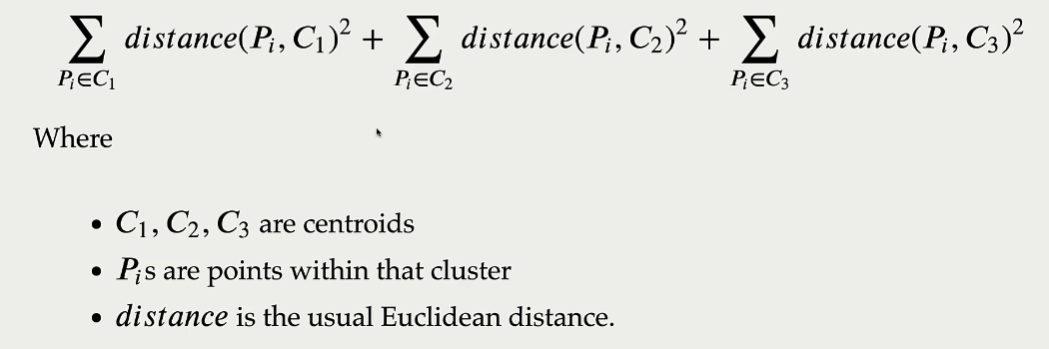
Sum of intra cluster distances which is also known as interia.

Run below code with for loop and plot line.

kmeans = KMeans(n\_clusters=k)

# Fit the model to the data and append the WCSS to the list

wcss.append(kmeans.fit(X).inertia\_)



### Package yellowbrick

model = Kmeans()

visual = **KElbowVisulizer**(model, k = (1, 10))

**visual** .fit(X)

visual.draw()

It will automatically draw cluster.

### Silhouette method

* The diffence between the average nearest-cluster distance nd average intra cluster distance for each sample and normalized by maximum value.

a(i) is the **mean distance** all other points in the same cluster

b(i) is the **mean distance** all other points in the nearest cluster

silhouette(i) = (b(i) - a(i)) / max(a(i), b(i))

* Measure the similarity of a data point to its own cluster versus the other clusters.
* Not dependent on the motion of cluster centres.
* Calculated using the mean intra clusteirng distance and mean nearest cluster distance for each sample.
* Range -1 to 1
* 1 indicating a high level of separation,
* -1 indicating a low level of separation.
* O mean overlapping
* **Highest average silhouette score is prefered**.

from Yellowbrick.cluster import SilhouetteVisualizer

model = Kmeans(3, random\_state = 42)

visualizer = SilhouetteVisualizer(model, colors=”yellobrick”)

**visualizer**.fit(x)

visualizer.show()

## Kmeans Problem

* unable to handle the complex cluster of the data
* scable variant is **MiniBatchMeans**
* predefine number of cluster
* not easily interpretable
* consider outliers as part of the cluster

# DBSCAN(updated version HDBSCAN)

Density based clustering algorithms

No need to specify number of cluster in advance

Able capture complex shape

## Hyperparameter

* Eps :

1. maximum distance between the points.
2. Points that closer have same eps.

* minimum\_samples :
  1. min number of the points needed to form the cluster .
  2. If a point has less then min\_samples points within the distance it is cosidered as the outliers.
  3. Increase the value we are creating more noise points.

## Points

Core points : at least min sample points

Border points : fewer than the min samples

Noise points : no cluster . less than min samples points within distance of eps.

## Cons

* Can not predict new example
* Needs to tune two non obvious parameter.

# Hierarchical Clustering

* Start each point as cluster
* Greedily add near by
* Repeat until reach one cluster
* Visulization using dendogram
* X- axis examples and y-axis is number of cluster

## Linkage criteria

Find similarity between the clusters.

Some examples

* Single linkage(minimum distance)

Merge two clusters that have smallest minimum distance between all their points

* Average linkage(average distance)
* Complete linkage(maximum distance)
* Ward linkage

# Dimensionality reduction technique:

Higher dimensional into lower dimensional

* PCA Pricipal component analysis : lower dim by findig the direction of high variance
* Singluar value decompoistion : decompse data matrix into the product of three matrics.
* t-distributed stochastic neighbour embedding(t-SNE) : higher dim to lower dim by preserving pairwise distances between them.
* Autoencoders
* TurncatedSVD or latent semantic analysis
* Non negative matrix factorization.

## PCA

lower dim by findig the direction of high variance

* high variance means low error

Summarize the data by finding best linear combinations of features.

First comp has the most most info then 2nd and so on.

𝐄𝐱𝐩𝐥𝐚𝐢𝐧𝐚𝐛𝐥𝐞 𝐀𝐈: 𝟏𝟎 𝐓𝐨𝐩 𝐏𝐲𝐭𝐡𝐨𝐧 𝐋𝐢𝐛𝐫𝐚𝐫𝐢𝐞𝐬 𝐟𝐨𝐫 𝐃𝐞𝐦𝐲𝐬𝐭𝐢𝐟𝐲𝐢𝐧𝐠 𝐘𝐨𝐮𝐫 𝐌𝐨𝐝𝐞𝐥'𝐬 𝐃𝐞𝐜𝐢𝐬𝐢𝐨𝐧𝐬

XAI is artificial intelligence that allows humans to understand the results and decision-making processes of the model or system.

𝟛 𝕊𝕥𝕒𝕘𝕖𝕤 𝕠𝕗 𝔼𝕩𝕡𝕝𝕒𝕟𝕒𝕥𝕚𝕠𝕟:

➡️Pre-modeling Explainability

Explainable AI starts with explainable data with exploratory data analysis and clear, interpretable feature engineering. For example, in the healthcare industry, a doctor may use exploratory data analysis to understand the patient's medical history, symptoms, and test results before making a diagnosis.

➡️Modeling Explainability

When choosing a model for a particular problem, it is generally best to use the most interpretable model that still achieves good predictive results. For example, in the finance industry, an analyst may use a decision tree model to predict loan defaults because it is easy to understand and interpret the decision making process.

➡️Post-model explainability

This includes techniques such as perturbation, where the effect of changing a single variable on the model's output is analyzed such as SHAP values for after training. For example, in the retail industry, a store manager may use SHAP values to understand why certain items are selling well and others are not.

I found these 𝟏𝟎 𝐩𝐲𝐭𝐡𝐨𝐧 𝐥𝐢𝐛𝐫𝐚𝐫𝐢𝐞𝐬 𝐟𝐨𝐫 𝐀𝐈 𝐞𝐱𝐩𝐥𝐚𝐢𝐧𝐚𝐛𝐢𝐥𝐢𝐭𝐲:

📚SHAP (SHapley Additive exPlanations)

SHAP is a model agnostic and works by breaking down the contribution of each feature and attributing a score to each feature.

📚 LIME (Local Interpretable Model-agnostic Explanations)

LIME is another model agnostic method that works by approximating the behavior of the model locally around a specific prediction.

📚 Eli5

Eli5 is a library for debugging and explaining classifiers. It provides feature importance scores, as well as "reason codes" for scikit-learn, Keras, xgboost, LightGBM, CatBoost.

📚 Shapash

Shapash is a Python library which aims to make machine learning interpretable and understandable to everyone. Shapash provides several types of visualization with explicit labels.

📚 Anchors

Anchors is a method for generating human-interpretable rules that can be used to explain the predictions of a machine learning model.

📚 XAI (eXplainable AI)

XAI is a library for explaining and visualizing the predictions of machine learning models including feature importance scores, decision trees, and rule-based explanations.

📚 BreakDown

BreakDown is a tool that can be used to explain the predictions of linear models. It works by decomposing the model's output into the contribution of each input feature.

📚 interpret-text

interpret-text is a library for explaining the predictions of natural language processing models.

📚 iml (Interpretable Machine Learning)

iml currently contains the interface and IO code from the Shap project, and it will potentially also do the same for the Lime project.

📚 aix360 (AI Explainability 360)

aix360 includes a comprehensive set of algorithms that cover different dimensions

📚 OmniXAI

OmniXAI (short for Omni eXplainable AI), addresses several problems with interpreting judgments produced by machine learning models in practice.

# machine-learning-environments topics reference

https://github.com/nielsborie/machine-learning-environments