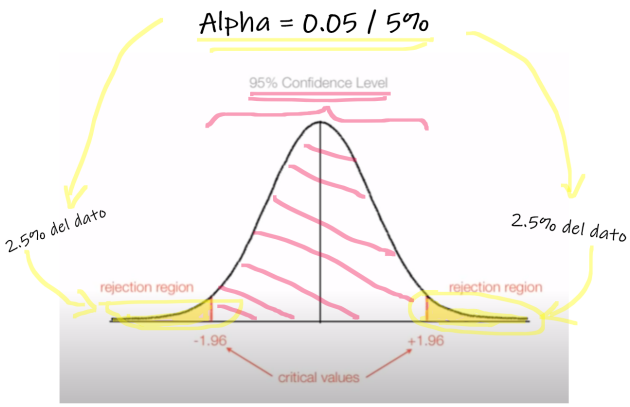
Pàgina web de machine learning i estadística amb python i R: https://www.cienciadedatos.net/ (https://www.cienciadedatos.net/)

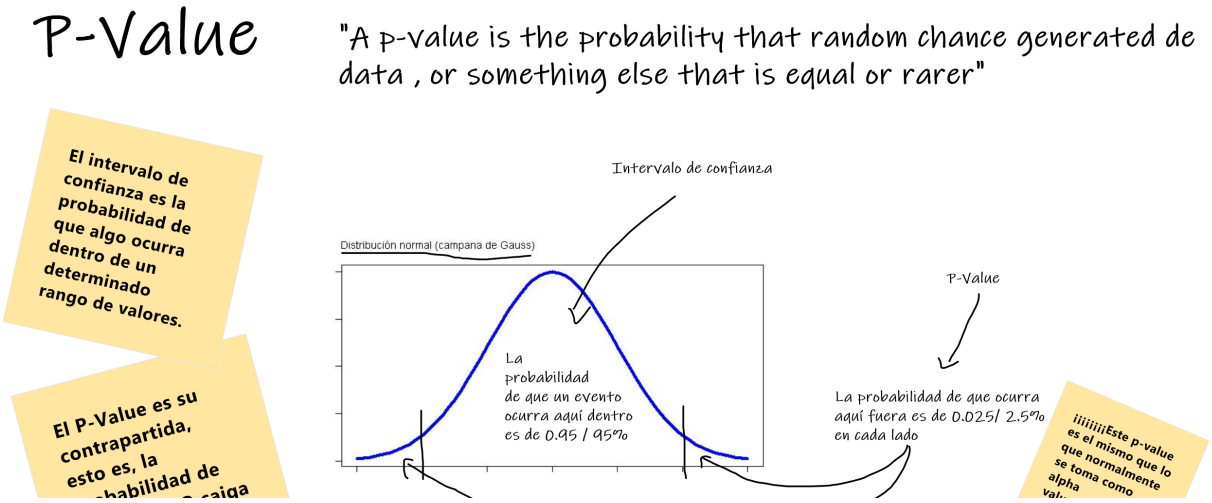
HYPOTHESIS TESTING

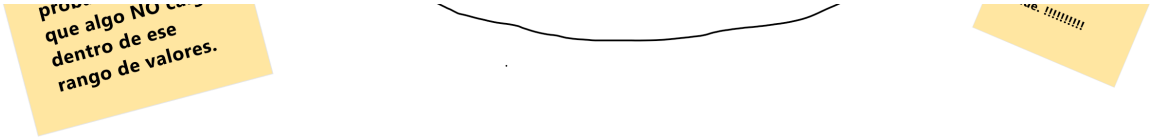
MUY BUENA WEB: https://www.cienciadedatos.net/estadistica-con-python.html (https://www.cienciadedatos.net/estadistica-con-python.html)

Alpha value

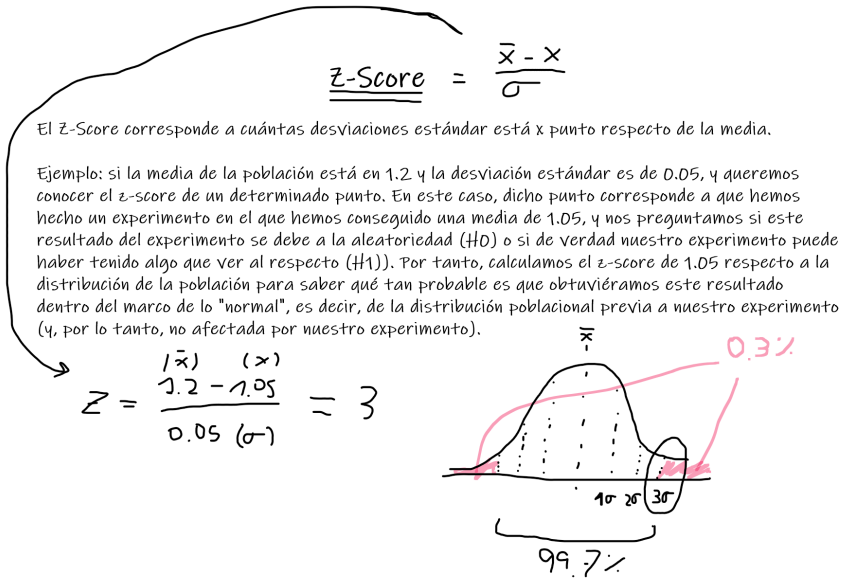


P-value





Z-Score



How to choose the right statistical test?

- https://www.youtube.com/watch?v=rulIUAN0U3w (https://www.youtube.com/watch?v=rulIUAN0U3w) (parametric tests)
- https://www.scribbr.com/statistics/statistical-tests/ (https://www.scribbr.com/statistics/statistical-tests/)

Statistical tests

https://machinelearningmastery.com/statistical-hypothesis-tests-in-python-cheat-sheet/ (https://machinelearningmastery.com/statistical-hypothesis-tests-in-python-cheat-sheet/)

1. Normality Tests: https://machinelearningmastery.com/a-gentle-introduction-to-normality-tests-in-python/ (https://machinelearningmastery.com/a-gentle-introduction-to-normality-tests-in-python/)

- Shapiro-Wilk Test: https://www.youtube.com/watch?v=eh9eYLBecWk&t=529s

(https://www.youtube.com/watch?v=eh9eYLBecWk&t=529s) // https://www.youtube.com/watch?v=HP6C32URXgg (https://www.youtube.com/watch?v=HP6C32URXgg)
- D'Agostino's K^2 Test
- Anderson-Darling Test

2. Correlation Tests: https://machinelearningmastery.com/how-to-use-correlation-to-understand-the-relationship-between-variables/ (https://machinelearningmastery.com/how-to-use-correlation-to-understand-the-relationship-between-variables/)

- Pearson's Correlation Coefficient
- Spearman's Rank Correlation
- Kendall's Rank Correlation
- Chi-Squared Test: https://machinelearningmastery.com/chi-squared-test-for-machine-learning/ (https://machinelearningmastery.com/chi-squared-test-for-machine-learning/)

3. Stationary Tests

- Augmented Dickey-Fuller
- Kwiatkowski-Phillips-Schmidt-Shin

4. Parametric tests

Regression tests: Regression tests are used to test cause-and-effect relationships. They look for the effect of one or more continuous variables on another variable.

- Simple Linear Regression: https://www.scribbr.com/statistics/simple-linear-regression/ (https://www.scribbr.com/statistics/simple-linear-regression/)
- Multiple Linear Regression: https://www.scribbr.com/statistics/multiple-linear-regression/ (https://www.scribbr.com/statistics/multiple-linear-regression/)
- Logistic Regression

Comparison tests: Comparison tests look for differences among group means.

- T-tests: https://www.scribbr.com/statistics/t-test/ (https://www.scribbr.com/statistics/t-test/)
- ANOVA: https://www.scribbr.com/statistics/one-way-anova/ (https://www.scribbr.com/statistics/one-way-anova/)

Correlation tests: Correlation tests check whether two variables are related without assuming cause-and-effect relationships.

- Pearson's r

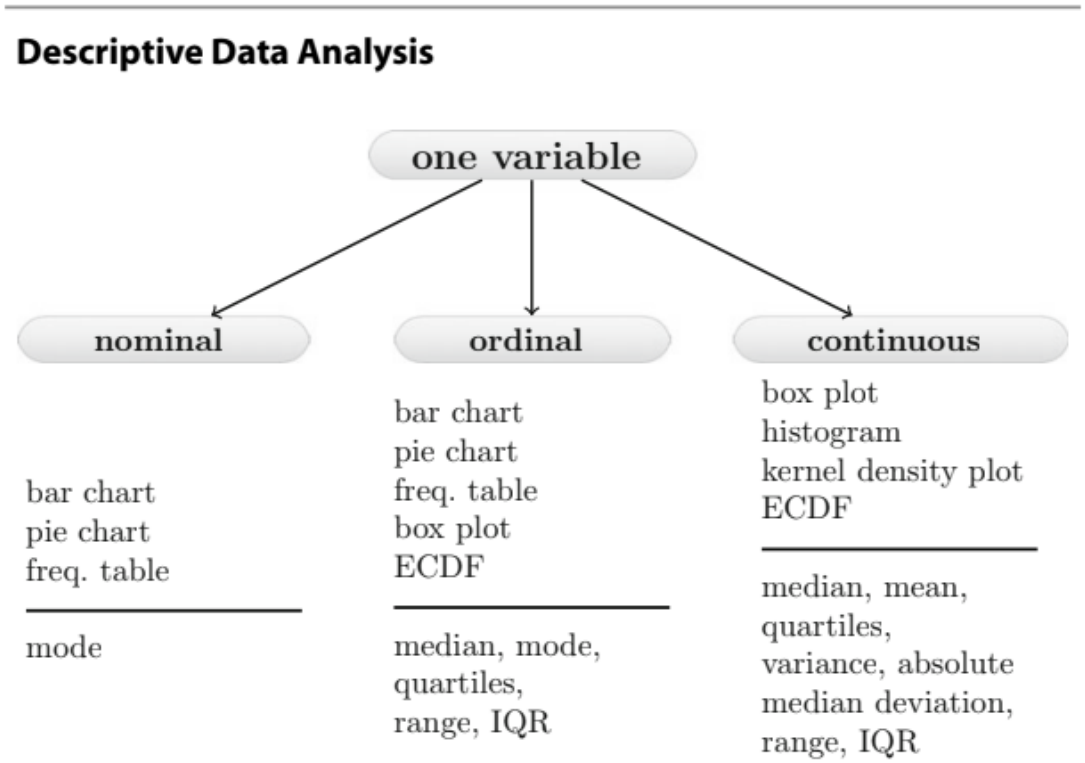
5. Nonparametric Statistical Hypothesis Tests

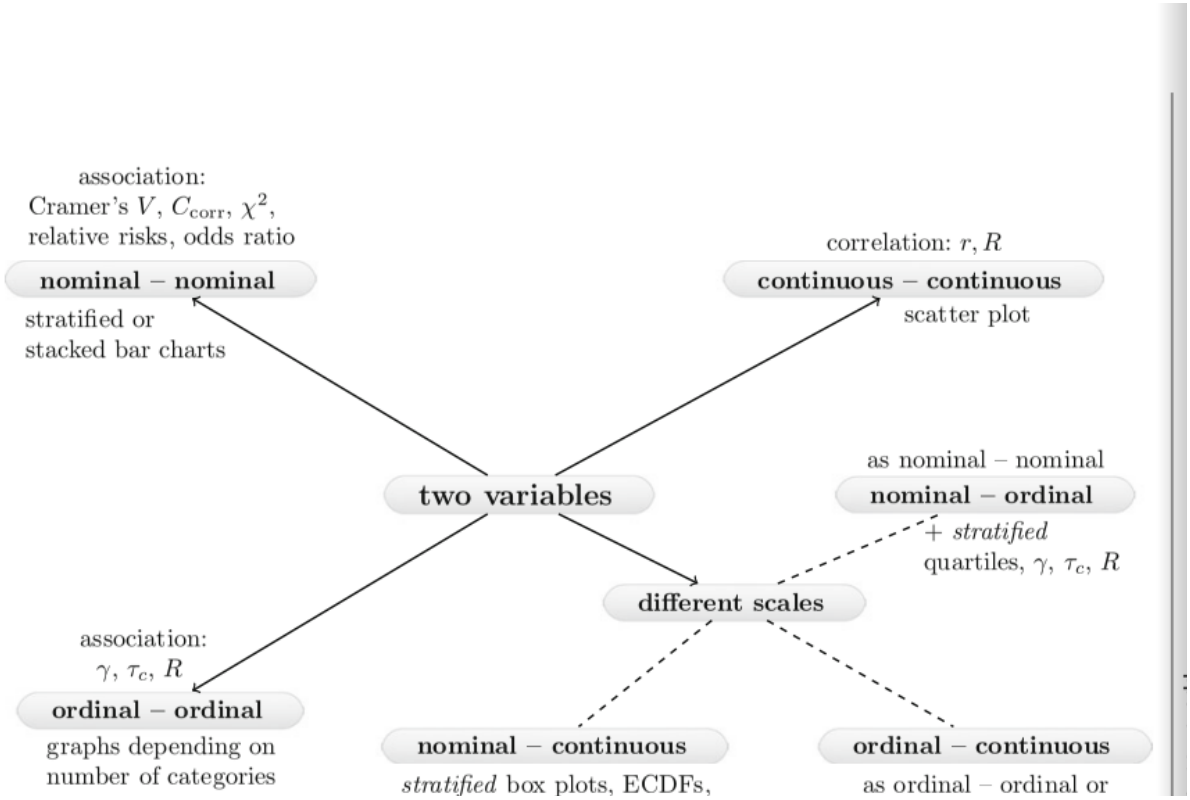
- Mann-Whitney U Test
- Wilcoxon Signed-Rank Test
- Kruskal-Wallis H Test: https://www.cienciadedatos.net/documentos/20_kruskal-wallis_test (https://www.cienciadedatos.net/documentos/20_kruskal-wallis_test)
- Friedman Test

Hypothesis testing with scikit-learn

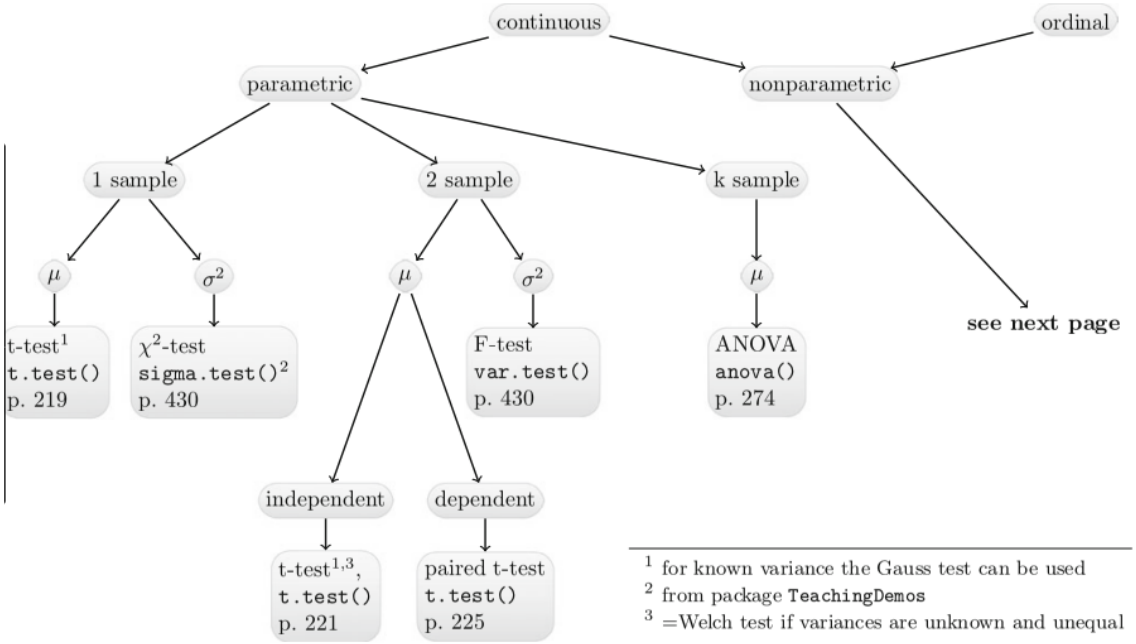
- https://www.youtube.com/watch?v=kd6zKBa9Rfk (https://www.youtube.com/watch?v=kd6zKBa9Rfk)

Esquemas

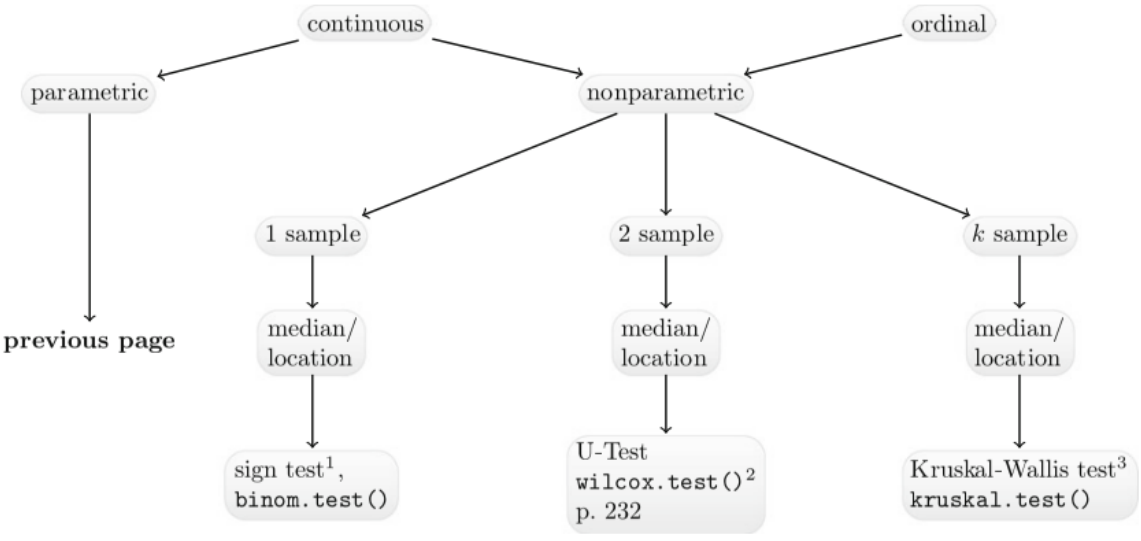




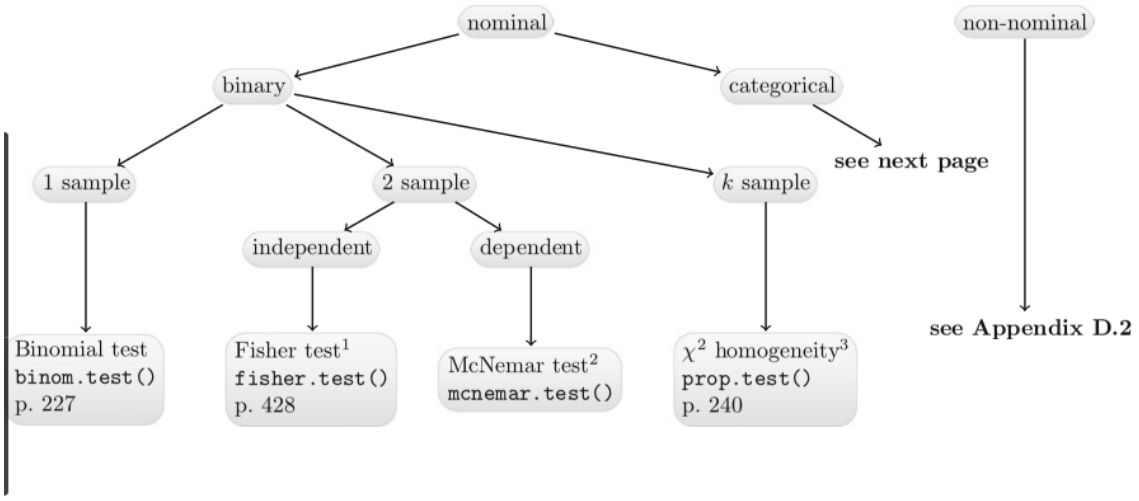
histograms, density plots, summary statistics	nominal – continuous



[1] for known variance the Gauss test can be used
[2] from package `TeachingDemos`
[3] =Welch test if variances are unknown and unequal

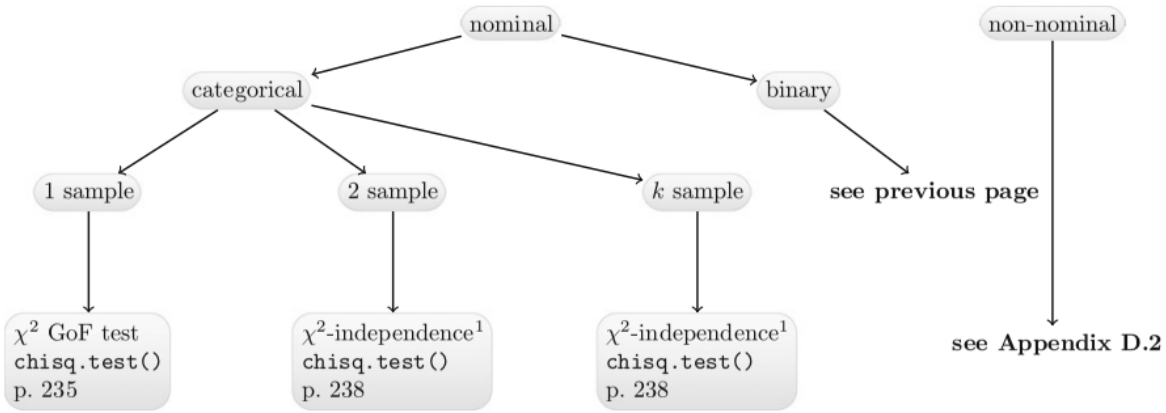


[1] not explained in this book; alternative: Mood's median test
[2] use option `paired=TRUE` for dependent data
[3] not explained in this book; use Friedman test for dependent data



[1] alternative: χ^2-independence test (`chisq.test()`)
(test decision equivalent to χ^2-homogeneity test)
[2] not explained in this book
[3] test decision equivalent to χ^2-independence test

Summary of Tests for Nominal Variables



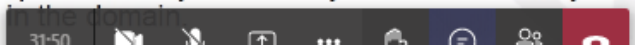
[1] test decision equivalent to χ^2-homogeneity test
(`prop.test`)

MASTERCLASS HYPOTHESIS TESTING

4. Preparació de dades

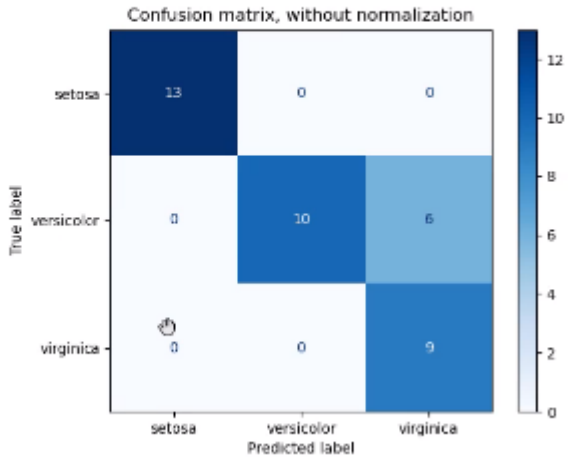
- Transformar atributs
  - **Centering**: Transform the data so that it has a mean of zero and a standard deviation of one. This is typically called data standardization.
  - **Scaling**: A standard scaling transformation is to map the data from the original scale to a scale between zero and one. This is typically called data normalization.
  - **Remove Skew**: Skewed data is data that has a distribution that is pushed to one side or the other (larger or smaller values) rather than being normally distributed. Some methods assume normally distributed data and can perform better if the skew is removed. Try replacing the attribute with the log, square root or inverse of the values.

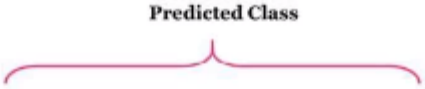
  - **Box-Cox**: A Box-Cox transform or family of transforms can be used to reliably adjust data to remove skew.
  - **Binning**: Numeric data can be made discrete by grouping values into bins. This is typically called data discretization. This process can be performed manually, although is more reliable if performed systematically and automatically using a heuristic that makes sense in the domain.



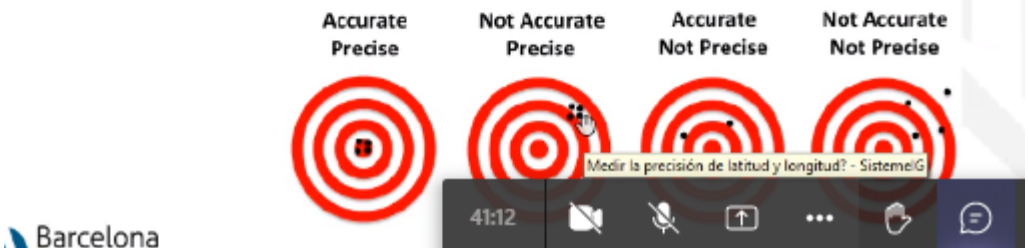
- Root mean square error
- Mean square error
- Lazy predict: ver qué algoritmo te funciona mejor para la predicción.
- Matriz de confusión

- Aprenentatge supervisat: Classificació



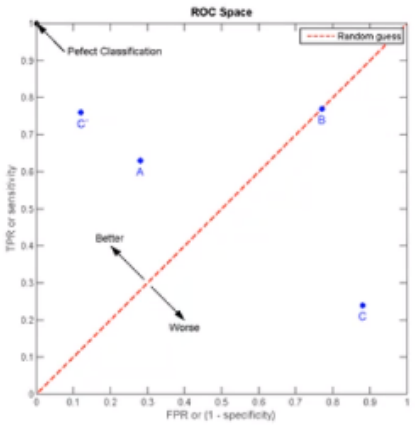


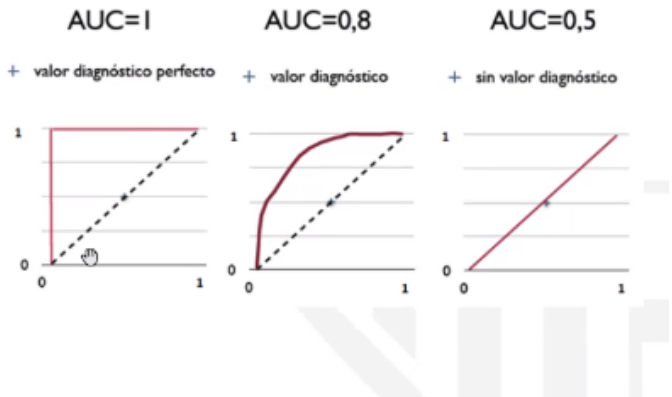
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	Negative Predictive Value $\frac{TN}{(TN+FN)}$	Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$



Barcelona

- • Area under the curve (AUC – ROC → Receiver Operating Characteristic)





- Servidor gratis: herokuapp.com

CORRELACIÓN, COVARIANZA Y FEATURE SCALING

Normalizar, estandarizar, transformar, etc. los datos para poder utilizarlos en modelos de Machine Learning.

Improve Model Accuracy with Data Pre-Processing

https://machinelearningmastery.com/improve-model-accuracy-with-data-pre-processing/

(https://machinelearningmastery.com/improve-model-accuracy-with-data-pre-processing/)

- Add attributes to your data
- Remove attributes from your data
- Transform attributes in your data

ADD

- Dummy Attributes: Categorical attributes can be converted into n-binary attributes, where n is the number of categories (or levels) that the attribute has. These denormalized or decomposed attributes are known as dummy attributes or dummy variables.
- Transformed Attribute: A transformed variation of an attribute can be added to the dataset in order to allow a linear method to exploit possible linear and non-linear relationships between attributes. Simple transforms like log, square and square root can be used.
- Missing Data: Attributes with missing data can have that missing data imputed using a reliable method, such as k-nearest neighbors.

Remove Data Attributes

Some methods perform poorly with redundant or duplicate attributes. You can get a boost in model accuracy by removing attributes from your data.

- Projection: Training data can be projected into lower dimensional spaces, but still characterize the inherent relationships in the data. A popular approach is Principal Component Analysis (PCA) where the principal components found by the method can be taken as a reduced set of input attributes.
- Spatial Sign: A spatial sign projection of the data will transform data onto the surface of a multidimensional sphere. The results can be used to highlight the existence of outliers that can be modified or removed from the data.
- Correlated Attributes: Some algorithms degrade in importance with the existence of highly correlated attributes. Pairwise attributes with high correlation can be identified and the most correlated attributes can be removed from the data.

Transform Data Attributes

Transformations of training data can reduce the skewness of data as well as the prominence of outliers in the data. Many models expect data to be transformed before you can apply the algorithm.

- Centering: Transform the data so that it has a mean of zero and a standard deviation of one. This is typically called data standardization.
- Scaling: A standard scaling transformation is to map the data from the original scale to a scale between zero and one. This is typically called data normalization.
- Remove Skew: Skewed data is data that has a distribution that is pushed to one side or the other (larger or smaller values) rather than being normally distributed. Some methods assume normally distributed data and can perform better if the skew is removed. Try replacing the attribute with the log, square root or inverse of the values.
- Box-Cox: A Box-Cox transform or family of transforms can be used to reliably adjust data to

remove skew.
- Binning: Numeric data can be made discrete by grouping values into bins. This is typically called data discretization. This process can be performed manually, although is more reliable if performed systematically and automatically using a heuristic that makes sense in the domain.

https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/ (https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/)

FEATURE ENGINEERING

Two steps:

- Feature transformation:

A) **Changing the scale of a variable** from original scale to scale between zero and one. This is known as data normalization. For example: If a data set has 1st variable in meter, 2nd in centi-meter and 3rd in kilo-meter, in such case, before applying any algorithm, we must normalize these variable in same scale.

B) Some algorithms works well with normally distributed data. Therefore, we must remove skewness of variable(s). There are methods like **log, square root or inverse of the values** to remove skewness.

https://towardsdatascience.com/top-3-methods-for-handling-skewed-data-1334e0debf45 (https://towardsdatascience.com/top-3-methods-for-handling-skewed-data-1334e0debf45) https://rk-journal.medium.com/log-transformation-how-it-can-transform-ml-model-performance-cd3ea944dfc7 (https://rk-journal.medium.com/log-transformation-how-it-can-transform-ml-model-performance-cd3ea944dfc7) https://stackoverflow.com/questions/21752989/numpy-efficiently-avoid-0s-when-taking-logmatrix (https://stackoverflow.com/questions/21752989/numpy-efficiently-avoid-0s-when-taking-logmatrix)

C) Some times, **creating bins of numeric data** works well, since it handles the outlier values also. Numeric data can be made discrete by grouping values into bins. This is known as data discretization.

- Feature Creation:

Deriving new variable(s) from existing variables is known as feature creation.

FEATURE SCALING

When to Normalize and Standardize (Feature Scaling)

https://machinelearningmastery.com/scale-machine-learning-data-scratch-python/ (https://machinelearningmastery.com/scale-machine-learning-data-scratch-python/)

Standardization is a scaling technique that **assumes your data conforms to a normal distribution**. If a given data attribute is normal or close to normal, this is probably the scaling method to use. It is good practice to record the summary statistics used in the standardization

process, so that you can apply them when standardizing data in the future that you may want to use with your model.

----------> *ESTANDARIZAR ES BÁSICAMENTE PASAR DOS VARIABLES O MÁS EN ESCALAS DIFERENTES A UNA MISMA ESCALA PARA PODERLAS COMPARAR*

Normalization is a scaling technique that **does not assume any specific distribution**. If your data is not normally distributed, consider normalizing it prior to applying your machine learning algorithm. It is good practice to record the minimum and maximum values for each column used in the normalization process, again, in case you need to normalize new data in the future to be used with your model.

----------> *NORMALIZAR ES BÁSICAMENTE PASAR DOS VARIABLES O MÁS A UNA ESCALA ENTRE -1/+1*

- Normalization that permits a configurable range, such as -1 to 1 and more.
- Standardization that permits a configurable spread, such as 1, 2 or more standard deviations from the mean.
- Exponential transforms such as logarithm, square root and exponents.
- Power transforms such as box-cox for fixing the skew in normally distributed data.

Scaling data is the process of increasing or decreasing the magnitude according to a fixed ratio, in simpler words you change the size but not the shape of the data. It helps improve the performance of the model and reducing the values/models from varying widely.

Centring

$$X_c = X - \bar{X}$$

Standardization: sklearn.preprocessing StandardScaler

Beneficios de standarizar:

- Intercepts are interpreted as the estimate when all predictors are at their mean value.
- Coefficients are in units of standard deviations of the original predictors. This allows for direct comparison of the magnitude of impact between different predictors.
- Optimization methods (minimizing loss functions) are faster and more stable.
- It is required for regularization penalties where the magnitude of coefficients for different predictors must have the same meaning.
- In K-Nearest Neighbors methods it is necessary if you want features to contribute equally since these models use the distance between observations calculated from the features.
- K-means clustering is affected by the scale of the data and standardizing the features will prevent variables from dominating simply based on their scale.
- In logistic regression, neural networks, and support vector machines unscaled data can result in a disproportionate effect of some data points over others.

Normalization: sklearn.preprocessing MinMaxScaler

Typically standardization is preferred to min-max normalization. However, there are some applications where min-max scaling would be preferable:

- Neural networks often require their inputs to be bounded between 0 and 1.
- In images, for example, where pixels can only take on a specific range of RGB values, data may have to be normalized.

Mirar ejemplos en https://medium.com/@stallonejacob/data-science-scaling-of-data-in-python-ec7ad220b339 (https://medium.com/@stallonejacob/data-science-scaling-of-data-in-python-ec7ad220b339)

FEATURE SCALING WITH OUTLIERS

https://machinelearningmastery.com/robust-scaler-transforms-for-machine-learning/ (https://machinelearningmastery.com/robust-scaler-transforms-for-machine-learning/)
https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/ (https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/)

Dummy variables

- pd.get_dummies() --> pd.concat()

FEATURE SELECTION

Feature Selection is a process of **finding out the best subset of attributes which better explains the relationship of independent variables with target variable**. La idea se basa en seleccionar las variables (features) más determinantes con relación al target, es decir, aquellas variables que influyen en mayor medida nuestro objeto de estudio.

- Domain Knowledge: Based on domain experience, we select feature(s) which may have higher impact on target variable.
- Visualization: As the name suggests, it helps to visualize the relationship between variables, which makes your variable selection process easier.
- Statistical Parameters. **PCA**: It helps to represent training data into lower dimensional spaces, but still characterize the inherent relationships in the data. It is a type of dimensionality reduction technique. There are various methods to reduce the dimensions (features) of training data like factor analysis, low variance, higher correlation, backward/ forward feature selection and others. Básicamente se trata de reducir las dimensiones/variables del dataset para poder analizar las relaciones existentes entre variables de manera más clara (ejemplo iris dataset).

---

https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/#:~:text=There%20are%20two%20main%20types,into%20wrapper,%20filter%20and%20intrins (https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/#:~:text=There%20are%20two%20main%20types,into%20wrapper,%20filter%20and%20intrins

1. Feature selection techniques:

- Supervised: wrapper, filter and intrinsic.
- Unsupervised.

2. Filter-based feature selection methods use statistical measures to score the correlation or dependence between input variables that can be filtered to choose the most relevant features.
3. Statistical measures for feature selection must be carefully chosen based on the data type of the input variable and the output or response variable.

FEATURE SELECTION METHODS

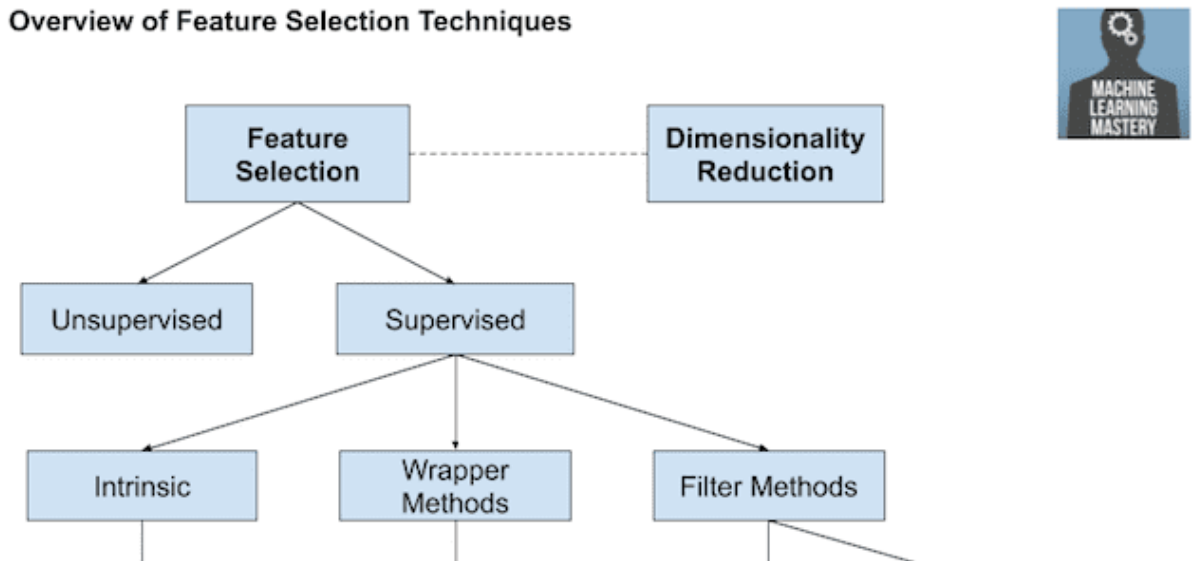
Feature selection methods are intended to reduce the number of input variables to those that are believed to be most useful to a model in order to predict the target variable.

One way to think about feature selection methods are in terms of supervised and unsupervised methods. **The difference has to do with whether features are selected based on the target variable or not**. Unsupervised feature selection techniques ignores the target variable, such as methods that remove redundant variables using correlation. Supervised feature selection techniques use the target variable, such as methods that remove irrelevant variables.

Wrapper feature selection methods create many models with different subsets of input features and select those features that result in the best performing model according to a performance metric. RFE is a good example of a wrapper feature selection method (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html)).

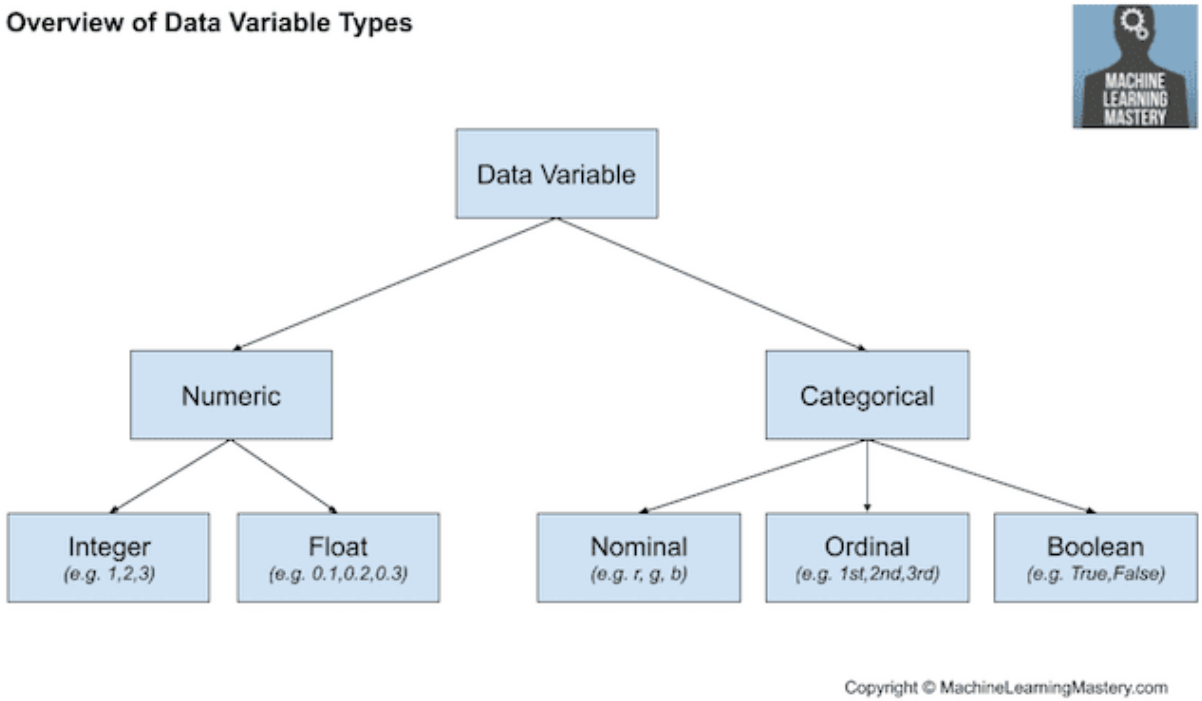
Filter feature selection methods use statistical techniques to evaluate the relationship between each input variable and the target variable, and these scores are used as the basis to choose (filter) those input variables that will be used in the model.

Finally, there are some machine learning algorithms that perform feature selection automatically as part of learning the model. We might refer to these techniques as **intrinsic** feature selection methods. This includes algorithms such as penalized regression models like Lasso and decision trees, including ensembles of decision trees like random forest.



FILTER METHODS

Based on data type:



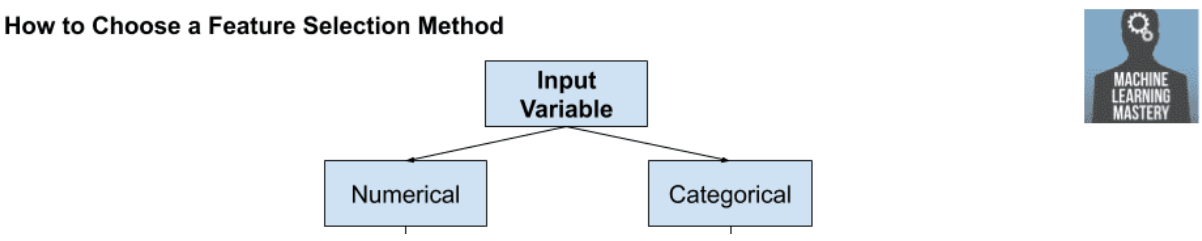
We will consider two broad categories of variable types: numerical and categorical; also, the two main groups of variables to consider: input and output.

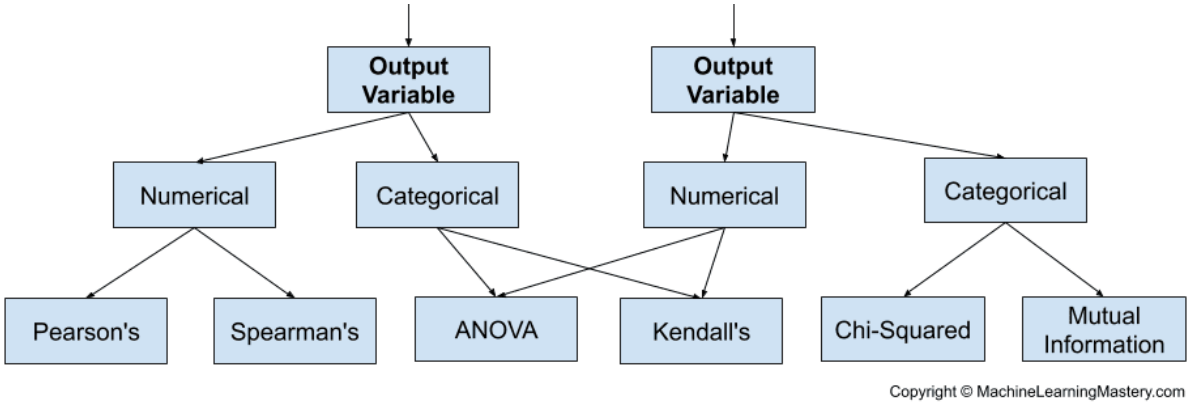
Input variables are those that are provided as input to a model. In feature selection, it is this group of variables that we wish to reduce in size. Output variables are those for which a model is intended to predict, often called the response variable.

The type of response variable typically indicates the type of predictive modeling problem being performed. For example, a numerical output variable indicates a regression predictive modeling problem, and a categorical output variable indicates a classification predictive modeling problem.

- Numerical Output: Regression predictive modeling problem.
- Categorical Output: Classification predictive modeling problem.

CHOOSING FEATURE SELECTION METHOD





Correlation Statistics

The scikit-learn library provides an implementation of most of the useful statistical measures.

- Pearson's Correlation Coefficient: f_regression()
- ANOVA: f_classif()
- Chi-Squared: chi2()
- Mutual Information: mutual_info_classif() and mutual_info_regression()
- Also, the SciPy library provides an implementation of many more statistics, such as Kendall's tau (kendalltau) and Spearman's rank correlation (spearmanr).

Selection method

The scikit-learn library also provides many different filtering methods once statistics have been calculated for each input variable with the target.

Two of the more popular methods include:

- Select the top k variables: SelectKBest
- Select the top percentile variables: SelectPercentile

---> EJEMPLOS EN EL LINK

ENSEMBLE METHODS

This is the most common approach found majorly in winning solutions of Data science competitions. This technique simply combines the result of multiple weak models and produce better results. This can be achieved through many ways:

- Bagging (Bootstrap Aggregating)
- Boosting

Más info: https://www.analyticsvidhya.com/blog/2015/08/introduction-ensemble-learning/ (https://www.analyticsvidhya.com/blog/2015/08/introduction-ensemble-learning/)

NOTA

Sometimes, the improvement in model's accuracy can be due to **over-fitting** too. To find the right answer of this question, we must use **cross validation technique**. Cross Validation is one of the

most important concepts in data modeling. It says, try to leave a sample on which you do not train the model and test the model on this sample before finalizing the model.

Más info: https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/ (https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/)

PCA

Linear Regression

Linear regression is the process of finding the linear function that is as close as possible to the actual relationship between features. In other words, you determine the linear function that best describes the association between the features. This linear function is also called the regression line.

You can use scipy.stats.linregress() to perform linear regression for two arrays of the same length:

```
>>> result = scipy.stats.linregress(x, y)
>>> result.slope
7.4363636363636365
>>> result.intercept
-85.92727272727274
>>> result.rvalue
0.7586402890911869
>>> result.pvalue
0.010964341301680825
>>> result.stderr
2.257878767543913
```

- .slope: the slope of the regression line
- .intercept: the intercept of the regression line
- .pvalue: the p-value
- .stderr: the standard error of the estimated gradient

FEATURE ENCODING

https://towardsdatascience.com/all-about-categorical-variable-encoding-305f3361fd02 (https://towardsdatascience.com/all-about-categorical-variable-encoding-305f3361fd02)

Binary Encoding - Yes or no

Nominal Encoding: — Where Order of data does not matter

- One Hot Encoding
- Freq. Encoding (https://www.kaggle.com/bhavikapanara/frequency-encoding (https://www.kaggle.com/bhavikapanara/frequency-encoding))

```
# Frecuency encoding
geo=['UniqueCarrier','Origin','Dest','Skyway']
for categoria in geo:
    # mapeo frecuencia de aparición de variables en train dataset
    cat_mapping=X.groupby(categoria).size()/len(X)*100
    # Imputo frecuencias en train y test dataset
    X[categoria]=X[categoria].map(cat_mapping)

```

- Mean Encoding

Ordinal Encoding: — Where Order of data matters

- Ordinal Encoding

Cyclical values (hours, mins, months, etc.): http://blog.davidkaleko.com/feature-engineering-cyclical-features.html (http://blog.davidkaleko.com/feature-engineering-cyclical-features.html) https://www.kaggle.com/avanwyk/encoding-cyclical-features-for-deep-learning (https://www.kaggle.com/avanwyk/encoding-cyclical-features-for-deep-learning)

INTRO MACHINE LEARNING

- Intro ML: https://www.w3schools.com/python/python_ml_linear_regression.asp (https://www.w3schools.com/python/python_ml_linear_regression.asp)
- Random module reference: https://www.w3schools.com/python/module_random.asp (https://www.w3schools.com/python/module_random.asp)
- Request module: https://www.w3schools.com/python/module_requests.asp (https://www.w3schools.com/python/module_requests.asp)
- Statistics module: https://www.w3schools.com/python/module_statistics.asp (https://www.w3schools.com/python/module_statistics.asp)
- Math module: https://www.w3schools.com/python/module_math.asp (https://www.w3schools.com/python/module_math.asp)
- Complex math module: https://www.w3schools.com/python/module_cmath.asp (https://www.w3schools.com/python/module_cmath.asp)
- Intro Scikit-learn: https://scikit-learn.org/stable/tutorial/basic/tutorial.html (https://scikit-learn.org/stable/tutorial/basic/tutorial.html)
- **Scikit-learn tutorial prepocessing package**: https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)

TRAIN AND TEST

INPUT (X) AND OUTPUT (y) DATA

https://towardsdatascience.com/the-abc-of-machine-learning-ea85685489ef

(https://towardsdatascience.com/the-abc-of-machine-learning-ea85685489ef)

TRAIN AND TEST

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)
https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/
(https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/)

The objective is to estimate the performance of the machine learning model on new data: data not used to train the model. The train-test procedure is appropriate when there is a sufficiently large dataset available. It means that there is enough data to split the dataset into train and test datasets and each of the train and test datasets are suitable representations of the problem domain. A suitable representation of the problem domain means that there are enough records to cover all common cases and most uncommon cases in the domain. If you have insufficient data, then a suitable alternate model evaluation procedure would be the **k-fold cross-validation** procedure.

You must choose a split percentage that meets your project's objectives with considerations that include:

- Computational cost in training the model.
- Computational cost in evaluating the model.
- Training set representativeness.
- Test set representativeness.

Common split percentages include:

- Train: 80%, Test: 20%
- Train: 67%, Test: 33%
- Train: 50%, Test: 50%

Código:

```
    train, test = train_test_split(dataset, ...)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
...)
    X_train, X_test, y_train, y_test = train_test_split(X, y, tes
t_size=0.33)
```

- *train_size* is the number that defines the size of the training set. If you provide a float, then it must be between 0.0 and 1.0 and will define the share of the dataset used for testing. If you provide an int, then it will represent the total number of the training samples. The default value is None.
- *test_size* is the number that defines the size of the test set. It's very similar to train_size. You should provide either train_size or test_size. If neither is given, then the default share of the dataset that will be used for testing is 0.25, or 25 percent.
- *random_state* is the object that controls randomization during splitting. It can be either an int or an instance of RandomState. The default value is None.
- *shuffle* is the Boolean object (True by default) that determines whether to shuffle the dataset

before applying the split.
- *stratify* is an array-like object that, if not None, determines how to use a stratified split.

Train-Test Split for Classification (Random Forest)

First, the loaded dataset must be split into input and output components (entre valores y variables)

```
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
print(X.shape, y.shape)
>>>(208, 60) (208,)
```

y, después, se separa entre train y test:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, tes
t_size=0.33, random_state=1)
```

fit:

```
model = RandomForestClassifier(random_state=1)
model.fit(X_train, y_train)
```

predict/evaluate:

```
# make predictions
yhat = model.predict(X_test)
# evaluate predictions
acc = accuracy_score(y_test, yhat)
print('Accuracy: %.3f' % acc)
```

Train-Test Split for Regression (Random Forest)

Lo mismo, solo que para evaluar la precisión utiliza el mean absolute error (en lugar de *accuracy_score*):

```
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

UNDERFITTING AND OVERFITTING

https://realpython.com/train-test-split-python-data/ (https://realpython.com/train-test-split-python-data/) **(Releer para los parámetros .coef y .score al aplicar regresion lineal)**

Splitting a dataset might also be important for detecting if your model suffers from one of two very common problems, called underfitting and overfitting:

- Underfitting is usually the consequence of a model being unable to encapsulate the relations among data. For example, this can happen when trying to represent nonlinear relations with a linear model. Underfitted models will likely have poor performance with both training and test

sets.

- Overfitting usually takes place when a model has an excessively complex structure and learns both the existing relations among data and noise. Such models often have bad generalization capabilities. Although they work well with training data, they usually yield poor

---

MACHINE LEARNING PROPIAMENTE

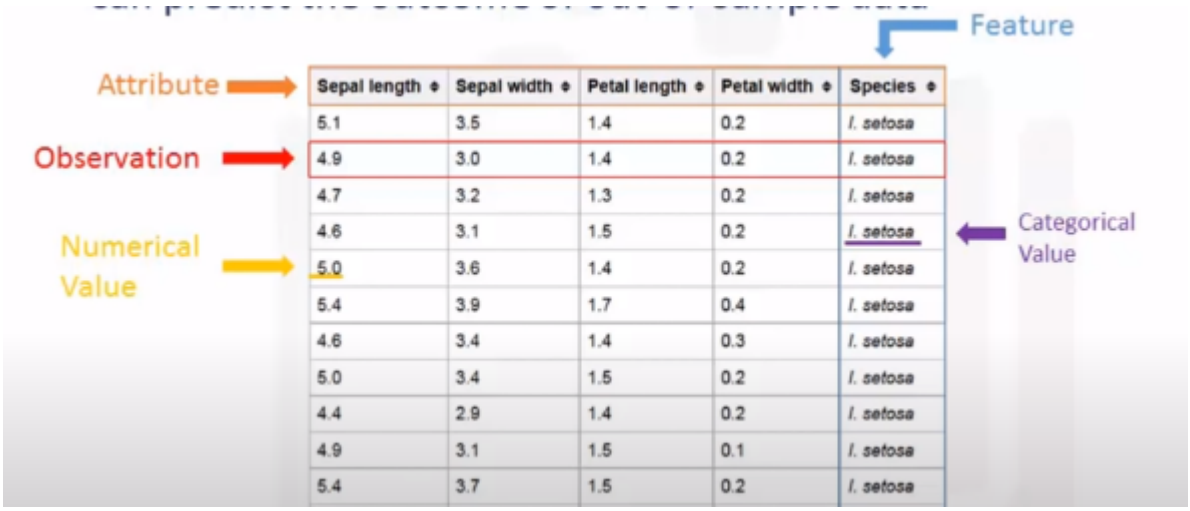
https://scikit-learn.org/stable/user_guide.html#user-guide (https://scikit-learn.org/stable/user_guide.html#user-guide)

APRENDIZAJE SUPERVISADO: REGRESIONES

Supervisado vs no supervisado

https://www.youtube.com/watch?v=cfj6yaYE86U&feature=emb_logo (https://www.youtube.com/watch?v=cfj6yaYE86U&feature=emb_logo)

- Atributos: las diferentes variables.
- Feature: una variable y todas sus observaciones
- Observación: cada caso en concreto recogido en el data set, una fila.



Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	*I. setosa*
4.9	3.0	1.4	0.2	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*
4.6	3.1	1.5	0.2	*I. setosa*
5.0	3.6	1.4	0.2	*I. setosa*
5.4	3.9	1.7	0.4	*I. setosa*
4.6	3.4	1.4	0.3	*I. setosa*
5.0	3.4	1.5	0.2	*I. setosa*
4.4	2.9	1.4	0.2	*I. setosa*
4.9	3.1	1.5	0.1	*I. setosa*
5.4	3.7	1.5	0.2	*I. setosa*

Regression

https://realpython.com/linear-regression-in-python/ (https://realpython.com/linear-regression-in-python/)

Example. Data related to each employee represent one observation. The presumption is that the experience, education, role, and city are the independent features, while the salary depends on

them.

- The dependent features are called the dependent variables, outputs, or responses.
- The independent features are called the independent variables, inputs, or predictors.

Regression problems usually have one continuous and unbounded dependent variable. The inputs, however, can be continuous, discrete, or even categorical data such as gender, nationality, brand, and so on.

Typically, you need regression to answer whether and how some phenomenon influences the other or how several variables are related. For example, you can use it to determine if and to what extent the experience or gender impact salaries.

Linear Regression

When implementing linear regression of some dependent variable y on the set of independent variables $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the number of predictors, you assume a linear relationship between y and $\mathbf{x}$: $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \varepsilon$. This equation is the regression equation. $\beta_0, \beta_1, \ldots, \beta_r$ are the **regression coefficients**, and ε **is the random error**.

Linear regression calculates the **estimators** of the regression coefficients or simply the **predicted weights**, denoted with $b_0, b_1, \ldots, b_r$. They define the estimated regression function $f(\mathbf{x}) = b_0 + b_1 x_1 + \cdots + b_r x_r$. This function should capture the dependencies between the inputs and output sufficiently well.

The estimated or predicted response, $f(\mathbf{x}_i)$, for each observation $i = 1, \ldots, n$, should be as close as possible to the corresponding actual response y_i. The differences $y_i - f(\mathbf{x}_i)$ for all observations $i = 1, \ldots, n$, are called the **residuals**. Regression is about determining the best predicted weights, that is the weights corresponding to the smallest residuals.

To get the best weights, you usually minimize the sum of squared residuals (SSR) for all observations $i = 1, \ldots, n$: $\text{SSR} = \Sigma_i (y_i - f(\mathbf{x}_i))^2$. This approach is called the method of **ordinary least squares**.

The coefficient of determination, denoted as R^2, tells you which amount of variation in y can be explained by the dependence on $\mathbf{x}$ using the particular regression model. Larger R^2 indicates a better fit and means that the model can better explain the variation of the output with different inputs.

The value $R^2 = 1$ corresponds to SSR = 0, that is to the perfect fit since the values of predicted and actual responses fit completely to each other.

Simple Linear Regression

The estimated regression function (black line) has the equation $f(x) = b_0 + b_1 x$. Your goal is to calculate the optimal values of the predicted weights b_0 and b_1 that minimize SSR and determine the estimated regression function. The value of b_0, also called the **intercept**, shows the point where the estimated regression line crosses the y axis. It is the value of the estimated response $f(x)$ for $x = 0$. The value of b_1 determines the **slope** of the estimated regression line.

Multiple Linear Regression

Multiple or multivariate linear regression is a case of linear regression with two or more independent variables.

Polynomial Regression

Procedimiento (Común a los diferentes tipos de regresión)

Transform input data (only for polynomial regression)

As you've seen earlier, you need to include x^2 (and perhaps other terms) as additional features when implementing polynomial regression. For that reason, you should transform the input array x to contain the additional column(s) with the values of x^2 (and eventually more features).

```
    transformer = PolynomialFeatures(degree=2, include_bias=Fals
e)
    fit / transform
```

parameters to PolynomialFeatures:

- *degree* is an integer (2 by default) that represents the degree of the polynomial regression function.
- *interaction_only* is a Boolean (False by default) that decides whether to include only interaction features (True) or all features (False).
- *include_bias* is a Boolean (True by default) that decides whether to include the bias (intercept) column of ones (True) or not (False).

Parameters to LinearRegression:

```
 model = LinearRegression()
```

- *fit_intercept* is a Boolean (True by default) that decides whether to calculate the intercept b_0 (True) or consider it equal to zero (False).
- *normalize* is a Boolean (False by default) that decides whether to normalize the input variables (True) or not (False).
- *copy_X* is a Boolean (True by default) that decides whether to copy (True) or overwrite the input variables (False).
- *n*jobs_ is an integer or None (default) and represents the number of jobs used in parallel computation. None usually means one job and -1 to use all processors.

Checking results

- You can obtain the coefficient of determination (R^2) with .score() called on model. The arguments are also the predictor x and regressor y.
- The attributes of model are .intercept_, which represents the coefficient, b_0 and .coef_, which represents b_1. Ejemplo:

```
>>> print('intercept:', model.intercept_)
intercept: 5.633333333333329
>>> print('slope:', model.coef_)
slope: [0.54]
```

---> The value b_0 = 5.63 (approximately) illustrates that your model predicts the response 5.63 when x is zero. The value b_1 = 0.54 means that the predicted response rises by 0.54 when x is increased by one.

Predicting / Evaluating

- To obtain the predicted response, use .predict()

Advanced Linear Regression With statsmodels

https://towardsdatascience.com/simple-and-multiple-linear-regression-in-python-c928425168f9 (https://towardsdatascience.com/simple-and-multiple-linear-regression-in-python-c928425168f9)

1. Separar en dos dataframes las variables independientes (X) y el target (y)
2. Elegir las variables independientes que puedan resultar buenas a la hora de predecir la variablie independiente.

We need to choose variables that we think we'll be good predictors for the dependent variable — that can be done by checking the correlation(s) between variables, by plotting the data and searching visually for relationship, by conducting preliminary research on what variables are good predictors of y etc.

3. Escribir el código añadiendo la constante.

```
import statsmodels.api as sm # import statsmodels
X = df["RM"] ## X usually means our input variables (or indep
endent variables)
y = target["MEDV"] ## Y usually means our output/dependent va
riable
X = sm.add_constant(X) ## let's add an intercept (beta_0) to
our model

# Note the difference in argument order
model = sm.OLS(y, X).fit() ## sm.OLS(output, input)
predictions = model.predict(X)

# Print out the statistics
model.summary()
```

4. Interpretar los resultados

- Df Residuals: Related to the degrees of freedom. The number of values in the final calculation of a statistic that are free to vary.
- Coefficient: how much the predicted value increases as the independent value increases by 1.

- R squared: the percentage of variance our model explains
- Standard error: the standard deviation of the sampling distribution of a statistic, most commonly of the mean.
- the t scores and p-values for hypothesis test.
- y intercept

---

https://towardsdatascience.com/simple-and-multiple-linear-regression-in-python-c928425168f9 (https://towardsdatascience.com/simple-and-multiple-linear-regression-in-python-c928425168f9)

Linear Reg.

Equation Simple L. R.

```
Y = mX + b
```

Y is the dependent variable — or the variable we are trying to predict or estimate; X is the independent variable — the variable we are using to make predictions; m is the slope of the regression line. B is Y-intercept.

Equation Multiple LR

```
Y'i = b0 + b1X1i + b2X2i
```

The regression equation is pretty much the same as the simple regression equation, just with more variables.

Evaluating the model

https://towardsdatascience.com/what-are-the-best-metrics-to-evaluate-your-regression-model-418ca481755b (https://towardsdatascience.com/what-are-the-best-metrics-to-evaluate-your-regression-model-418ca481755b)

- R squared: how much variability in dependent variable can be explained by the model

$$R^2 = \frac{\text{Suma de cuadrados totales - Suma de cuadrados residuales}}{\text{Suma de cuadrados totales}} = 1 - \frac{\text{Suma de cuadrados residuales}}{\text{Suma de cuadrados totales}} = 1 - \frac{\sum(\hat{y}_i - y_i)^2}{\sum(y_i - \bar{y})^2}$$

- R sqrd adjusted: En los modelos lineales múltiples, cuantos más predictores se incluyan en el modelo, mayor es el valor de R2 . Esto es así ya que, por poco que sea, cada predictor va a explicar una parte de la variabilidad observada en y . Es por esto que R2 no puede utilizarse para comparar modelos con distinto número de predictores. R2ajustado introduce una penalización al valor de R2 por cada predictor que se añade al modelo. R2ajustado

permite encontrar el mejor modelo, aquel que consigue explicar mejor la variabilidad de y con el menor número de predictores, considerando así el problema de Overfitting (no contemplado por R squared).
- Mean Square Error / Root Mean Square Error: It gives you an absolute number on how much your predicted results deviate from the actual number. You cannot interpret many insights from one single result but it gives you a real number to compare against other model results and help you select the best regression model.
- Mean Absolute Error: Sum of the absolute value (not the squared) of error. Compare to MSE or RMSE, MAE is a more direct representation of sum of error terms. MSE gives larger penalization to big prediction error by square it while MAE treats all errors the same.

R Square/Adjusted R Square is better used to explain the model to other people because you can explain the number as a percentage of the output variability. MSE, RMSE, or MAE are better be used to compare performance between different regression models. However, it makes total sense to use MSE if the value is not too big and MAE if you do not want to penalize large prediction errors.

Requisitos para la regresión lineal

https://www.cienciadedatos.net/documentos/py10-regresion-lineal-python.html (https://www.cienciadedatos.net/documentos/py10-regresion-lineal-python.html)

- **No colinealidad entre las variables independientes**. Cuando se intenta establecer relaciones causa-efecto, la colinealidad puede llevar a conclusiones muy erróneas, haciendo creer que una variable es la causa cuando, en realidad, es otra la que está influenciando sobre ese predictor. Formas de verificar la colinealidad (tabla de correlaciones, calcular el VIF, mirar si el R2 es demasiado alto, etc.). En caso de encontrar colinealidad entre predictores, hay dos posibles soluciones. La primera es excluir uno de los predictores problemáticos intentando conservar el que, a juicio del investigador, está influyendo realmente en la variable respuesta. Esta medida no suele tener mucho impacto en el modelo en cuanto a su capacidad predictiva ya que, al existir colinealidad, la información que aporta uno de los predictores es redundante en presencia del otro. La segunda opción consiste en combinar las variables colineales en un único predictor, aunque con el riesgo de perder su interpretación.
- **Relación lineal entre los predictores e y**. Cada predictor numérico tiene que estar linealmente relacionado con la variable respuesta y mientras los demás predictores se mantienen constantes, de lo contrario no se deben introducir en el modelo. La forma más recomendable de comprobarlo es representando los residuos del modelo frente a cada uno de los predictores. Si la relación es lineal, los residuos se distribuyen de forma aleatoria en torno a cero. Estos análisis son solo aproximados, ya que no hay forma de saber si realmente la relación es lineal cuando el resto de predictores se mantienen constantes

VISUALIZACIÓN DE MODELOS DE REGRESIÓN

https://seaborn.pydata.org/tutorial/regression.html (https://seaborn.pydata.org/tutorial/regression.html) (MARAVILLA)

RANDOM FOREST

https://towardsdatascience.com/random-forest-in-python-24d0893d51c0 (https://towardsdatascience.com/random-forest-in-python-24d0893d51c0)

- Baseline (interesante pero todavía no sé si aplicable a otros ejemplos que no sean los del link)
- Feature importance: https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html (https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html)
- Choosing hyperparameters: https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74 (https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74)

NEURAL NETWORK

- Explicación parámetros y tips for practical use: https://scikit-learn.org/stable/modules/neural_networks_supervised.html (https://scikit-learn.org/stable/modules/neural_networks_supervised.html)

HOW TO CHOOSE MODEL

- https://www.youtube.com/watch?v=HdlDYng8g9s&list=PLBiSyPZ0731-udEYaWcg_RTdk5dNfn3ap&index=1 (https://www.youtube.com/watch?v=HdlDYng8g9s&list=PLBiSyPZ0731-udEYaWcg_RTdk5dNfn3ap&index=1)
- https://towardsdatascience.com/lazy-predict-fit-and-evaluate-all-the-models-from-scikit-learn-with-a-single-line-of-code-7fe510c7281 (https://towardsdatascience.com/lazy-predict-fit-and-evaluate-all-the-models-from-scikit-learn-with-a-single-line-of-code-7fe510c7281)

HOW TO CHOOSE PARAMS

- (Uso de GridSeach muy interesante) https://github.com/codebasics/py/blob/master/ML/15_gridsearch/15_grid_search.ipynb (https://github.com/codebasics/py/blob/master/ML/15_gridsearch/15_grid_search.ipynb)
- https://www.cienciadedatos.net/documentos/py11-calibrar-modelos-machine-learning.html (https://www.cienciadedatos.net/documentos/py11-calibrar-modelos-machine-learning.html)

CLASIFICACIÓN

BASELINE

```
    accuracy_score(delayed_flights.ArrDelay, np.ones(len(delayed_
flights)))
```

DECISION TREES

https://www.datacamp.com/community/tutorials/decision-tree-classification-python (https://www.datacamp.com/community/tutorials/decision-tree-classification-python)

Consideraciones previas

- The time complexity of decision trees is a function of the number of records and number of attributes in the given data.
- No paramétrico
- Decision trees can handle high dimensional data with good accuracy.

Optimizing Decision Tree Performance

- criterion : optional (default="gini") or Choose attribute selection measure: This parameter allows us to use the different-different attribute selection measure. Supported criteria are "gini" for the Gini index and "entropy" for the information gain.
- splitter : string, optional (default="best") or Split Strategy: This parameter allows us to choose the split strategy. Supported strategies are "best" to choose the best split and "random" to choose the best random split.
- **max_depth** : int or None, optional (default=None) or Maximum Depth of a Tree: The maximum depth of the tree. If None, then nodes are expanded until all the leaves contain less than min_samples_split samples. The higher value of maximum depth causes overfitting, and a lower value causes underfitting (Source).

Pros y contras

Pros

- Decision trees are easy to interpret and visualize.
- It can easily capture Non-linear patterns.
- It requires fewer data preprocessing from the user, for example, there is no need to normalize columns.
- It can be used for feature engineering such as predicting missing values, suitable for variable selection.
- The decision tree has no assumptions about distribution because of the non-parametric nature of the algorithm.

Cons

- Sensitive to noisy data. It can overfit noisy data.

- The small variation(or variance) in data can result in the different decision tree. This can be reduced by bagging and boosting algorithms.
- Decision trees are biased with imbalance dataset, so it is recommended that balance out the dataset before creating the decision tree.

Evaluate the model

https://towardsdatascience.com/how-to-best-evaluate-a-classification-model-2edb12bcc587 (https://towardsdatascience.com/how-to-best-evaluate-a-classification-model-2edb12bcc587)

Accuracy

```
from sklearn import metrics
print(metrics.accuracy_score(y_test, y_pred_class))
```

Assume we are creating a model to perform binary classification on a dataset with an unbalanced class distribution. 93% of data points are in class A and 7% in class B. We have a model that only predicts class A. It is hard to even call it a “model” because it predicts class A without any calculation. However, since 93% of the samples are in class A, the accuracy of our model is 93%. What if it is crucial to detect class B correctly and we cannot afford to misclassify any class B samples (i.e. cancer prediction)? In these cases, we need other metrics to evaluate our model.

Confusion Matrix

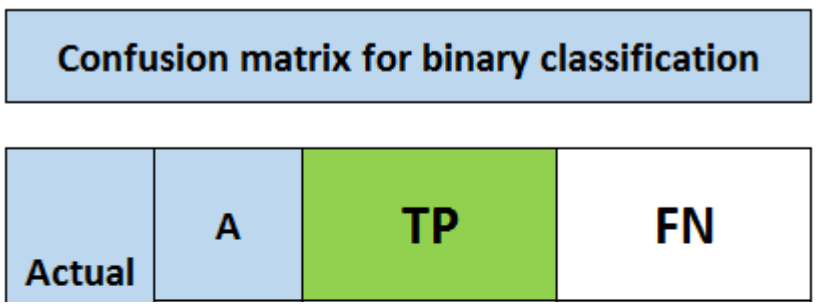
```
confusion = metrics.confusion_matrix(y_test, y_pred_class)
```

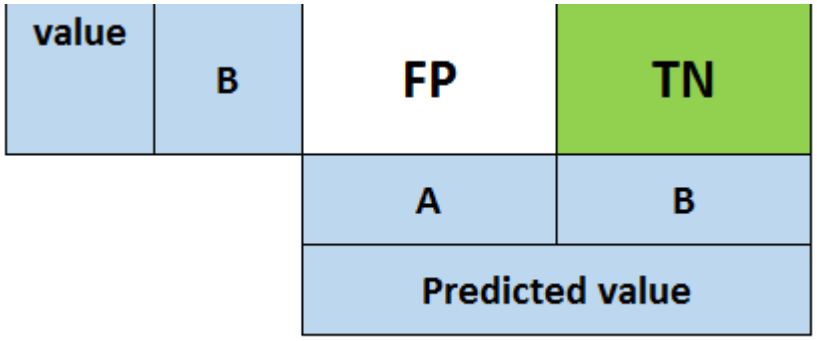
Confusion matrix goes deeper than classification accuracy by showing the correct and incorrect (i.e. true or false) predictions on each class. In case of a binary classification task, a confusion matrix is a 2x2 matrix. If there are three different classes, it is a 3x3 matrix and so on.

The key terms of confusion matrix are as follows:

- True positive (TP): Predicting positive class as positive (ok)
- False positive (FP): Predicting negative class as positive (not ok)
- False negative (FN): Predicting positive class as negative (not ok)
- True negative (TN): Predicting negative class as negative (ok)

```
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]
```





Precision, recall and F1

Precision es el porcentaje de True Positives sobre el total de valores positivos que ha detectado (True Positive + False Positive). Es interesante de cara a analizar, por ejemplo, el rendimiento de un modelo que detecta correos spam (antes preferimos que clasifique como mail un spam, que clasifique como spam un mail que no lo era).

```
print(metrics.precision_score(y_test, y_pred_class))
```

$$Precision = \frac{TP}{TP + FP}$$

Recall es el porcentaje de True Positives sobre el total de valores positivos reales (True Positives + False Negatives). Interesa de cara a analizar, por ejemplo, el rendimiento de un modelo que diagnostica cancer (preferimos que lo diagnostique sobre alguien que no lo tiene antes que no lo haga sobre alguien que sí lo tiene).

```
print(metrics.recall_score(y_test, y_pred_class))
```

$$Recall = \frac{TP}{TP + FN}$$

F1 score is the weighted average of precision and recall. F1 score is a more useful measure than accuracy for problems with uneven class distribution because it takes into account both false positive and false negatives. The best value for f1 score is 1 and the worst is 0.

```
f1 = f1_score(y_test, y_pred_prob)
```

Sensitivity and specificity

Sensitivity: When the actual value is positive, how often is the prediction correct?

```
sensitivity = TP / float(FN + TP)
```

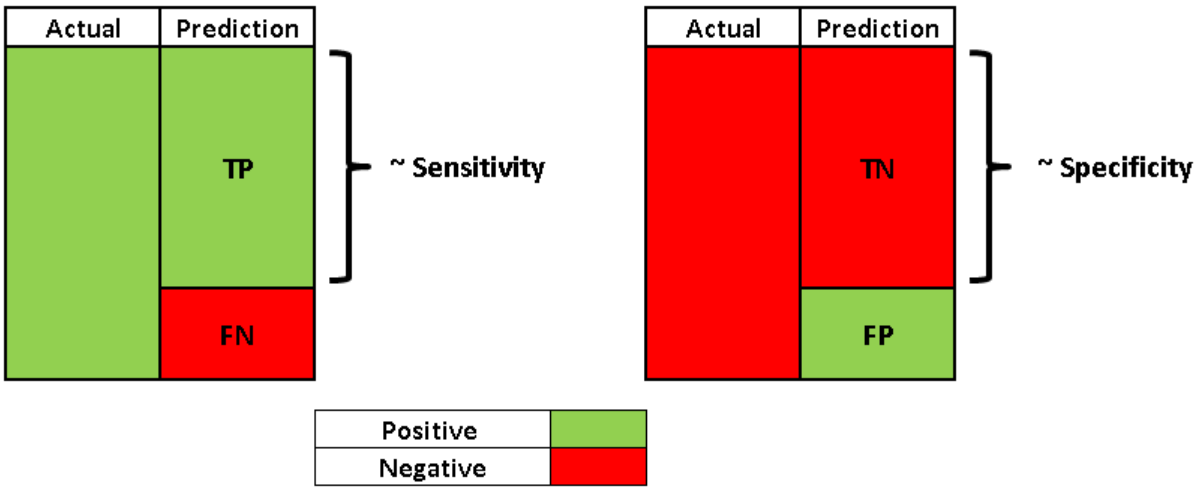
- Something we want to maximize

- How "sensitive" is the classifier to detecting positive instances?
- Also known as "True Positive Rate" or "Recall"
- TP / all positive (all positive = TP + FN)

Specificity: When the actual value is negative, how often is the prediction correct?

- Something we want to maximize
- How "specific" (or "selective") is the classifier in predicting positive instances?
- TN / all negative (all negative = TN + FP)

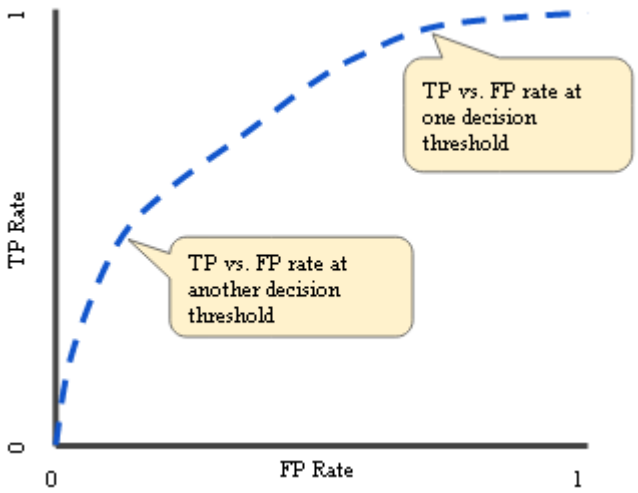
```
specificity = TN / (TN + FP)
```



--> Also: False positive rate

ROC Curve and AUC

ROC

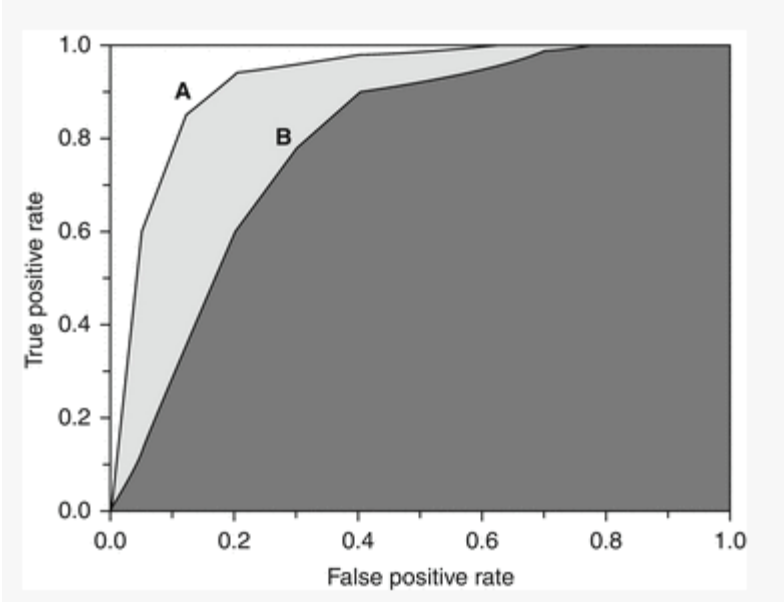


X axis of ROC curve is the true positive rate (sensitivity) and y axis of the ROC curve is the false positive rate (1- specificity). If the threshold is set to 0, the model predicts all samples as positive. In this case, TPR (sensitivity) is 1. However, FPR(1-specificity) is also 1 because there is no

negative prediction. If the threshold is set to 1, both TPR and FPR become 0. Hence, it is not a good choice to set the threshold to 0 or 1. If the threshold is set to 0, the model predicts all samples as positive. In this case, TPR (sensitivity) is 1. However, FPR(1-specificity) is also 1 because there is no negative prediction. If the threshold is set to 1, both TPR and FPR become 0. Hence, it is not a good choice to set the threshold to 0 or 1.

AUC

Instead of trying to find the optimum threshold value on ROC curve, we can use another metric called AUC (Area under the curve). AUC is the area under ROC curve between (0,0) and (1,1) which can be calculated using integral calculus. AUC basically aggregates the performance of the model at all threshold values. The best possible value of AUC is 1 which indicates a perfect classifier. The closer the AUC is to 1, the better the classifier is. In the figure below, classifier A is better than classifier B.




```
print(metrics.roc_auc_score(y_test, y_pred_prob))
```

- AUC is useful as a single number summary of classifier performance
- Higher value = better classifier
- If you randomly chose one positive and one negative observation, AUC represents the likelihood that your classifier will assign a higher predicted probability to the positive observation
- AUC is useful even when there is high class imbalance (unlike classification accuracy)

```
# calculate cross-validated AUC
from sklearn.cross_validation import cross_val_score
cross_val_score(logreg, X, y, cv=10, scoring='roc_auc').mean
()
```

K-NEAREST NEIGHBOURS

https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/

(https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/)

Pros

- It is extremely easy to implement
- As said earlier, it is lazy learning algorithm and therefore requires no training prior to making real time predictions. This makes the KNN algorithm much faster than other algorithms that require training e.g SVM, linear regression, etc.
- Since the algorithm requires no training before making predictions, new data can be added seamlessly.
- There are only two parameters required to implement KNN i.e. the value of K and the distance function (e.g. Euclidean or Manhattan etc.)

Cons

- The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate distance in each dimension.
- The KNN algorithm has a high prediction cost for large datasets. This is because in large datasets the cost of calculating distance between new point and each existing point becomes higher.
- Finally, the KNN algorithm doesn't work well with categorical features since it is difficult to find the distance between dimensions with categorical features.

LOGISTIC REGRESSION

Types of Logistic Regression:

- Binary Logistic Regression: The target variable has only two possible outcomes such as Spam or Not Spam, Cancer or No Cancer.
- Multinomial Logistic Regression: The target variable has three or more nominal categories such as predicting the type of Wine.
- Ordinal Logistic Regression: the target variable has three or more ordinal categories such as restaurant or product rating from 1 to 5.

Pros / cons

Advantages

Because of its efficient and straightforward nature, doesn't require high computation power, easy to implement, easily interpretable, used widely by data analyst and scientist. Also, it doesn't require scaling of features. Logistic regression provides a probability score for observations.

Disadvantages

Logistic regression is not able to handle a large number of categorical features/variables. It is vulnerable to overfitting. Also, can't solve the non-linear problem with the logistic regression that is why it requires a transformation of non-linear features. Logistic regression will not perform well with independent variables that are not correlated to the target variable and are very similar or

correlated to each other

SUPPORT VECTOR MACHINE

https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python (https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python)

SVM Kernels

The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form. SVM uses a technique called the kernel trick. Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space. In other words, you can say that it converts nonseparable problem to separable problems by adding more dimension to it. It is most useful in non-linear separation problem. Kernel trick helps you to build a more accurate classifier.

- Linear Kernel: A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.

K(x, xi) = sum(x * xi)

- Polynomial Kernel: A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.

K(x,xi) = 1 + sum(x * xi)^d

Where d is the degree of the polynomial. d=1 is similar to the linear transformation. The degree needs to be manually specified in the learning algorithm.

- Radial Basis Function Kernel: The Radial basis function kernel is a popular kernel function commonly used in support vector machine classification. RBF can map an input space in infinite dimensional space.

K(x,xi) = exp(-gamma * sum((x – xi^2))

Here gamma is a parameter, which ranges from 0 to 1. A higher value of gamma will perfectly fit the training dataset, which causes over-fitting. Gamma=0.1 is considered to be a good default value. The value of gamma needs to be manually specified in the learning algorithm.

Tuning Hyperparameters

- Kernel: The main function of the kernel is to transform the given dataset input data into the required form. There are various types of functions such as linear, polynomial, and radial basis function (RBF). Polynomial and RBF are useful for non-linear hyperplane. Polynomial and RBF kernels compute the separation line in the higher dimension. In some of the applications, it is suggested to use a more complex kernel to separate the classes that are curved or nonlinear. This transformation can lead to more accurate classifiers.
- Regularization: Regularization parameter in python's Scikit-learn **C** parameter used to maintain regularization. Here C is the penalty parameter, which represents misclassification or error term. The misclassification or error term tells the SVM optimization how much error is

bearable. This is how you can control the trade-off between decision boundary and misclassification term. A smaller value of C creates a small-margin hyperplane and a larger value of C creates a larger-margin hyperplane.

- Gamma: A lower value of Gamma will loosely fit the training dataset, whereas a higher value of gamma will exactly fit the training dataset, which causes over-fitting. In other words, you can say a low value of gamma considers only nearby points in calculating the separation line, while the a value of gamma considers all the data points in the calculation of the separation line.

Pros and cons

Advantages

SVM Classifiers offer good accuracy and perform faster prediction compared to Naïve Bayes algorithm. They also use less memory because they use a subset of training points in the decision phase. SVM works well with a clear margin of separation and with high dimensional space.

Disadvantages

SVM is not suitable for large datasets because of its high training time and it also takes more time in training compared to Naïve Bayes. It works poorly with overlapping classes and is also sensitive to the type of kernel used.

XGBOOST

https://www.datacamp.com/community/tutorials/xgboost-in-python (https://www.datacamp.com/community/tutorials/xgboost-in-python)

If you plan to use XGBoost on a dataset which has categorical features you may want to consider applying some encoding (like one-hot encoding) to such features before training the model. Also, if you have some missing values such as NA in the dataset you may or may not do a separate treatment for them, because XGBoost is capable of handling missing values internally.

1. Data Matrix

Now you will convert the dataset into an optimized data structure called Dmatrix that XGBoost supports and gives it acclaimed performance and efficiency gains.

```
data_dmatrix = xgb.DMatrix(data=X,label=y)
```

2. Building model

2.1. Hyperparameters

- learning_rate: step size shrinkage used to prevent overfitting. Range is [0,1]
- max_depth: determines how deeply each tree is allowed to grow during any boosting round.
- subsample: percentage of samples used per tree. Low value can lead to underfitting.
- colsample_bytree: percentage of features used per tree. High value can lead to overfitting.
- n_estimators: number of trees you want to build.
- objective: determines the loss function to be used like reg:linear for regression problems,

reg:logistic for classification problems with only decision, binary:logistic for classification problems with probability.

XGBoost also supports regularization parameters to penalize models as they become more complex and reduce them to simple (parsimonious) models.

- gamma: controls whether a given node will split based on the expected reduction in loss after the split. A higher value leads to fewer splits. Supported only for tree-based learners.
- alpha: L1 regularization on leaf weights. A large value leads to more regularization.
- lambda: L2 regularization on leaf weights and is smoother than L1 regularization.

2.2.

```
    #model
    xg_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_
bytree = 0.3, learning_rate = 0.1,
            max_depth = 5, alpha = 10, n_estimators = 10)

    #fit and predict
    xg_reg.fit(X_train,y_train)
    preds = xg_reg.predict(X_test)

    #MSE
    rmse = np.sqrt(mean_squared_error(y_test, preds))
```

3. Cross Validation

In order to build more robust models, it is common to do a k-fold cross validation where all the entries in the original training dataset are used for both training as well as validation. Also, each entry is used for validation just once. XGBoost supports k-fold cross validation via the cv() method. All you have to do is specify the nfolds parameter, which is the number of cross validation sets you want to build. More parameters:

- num_boost_round: denotes the number of trees you build (analogous to n_estimators)
- metrics: tells the evaluation metrics to be watched during CV
- as_pandas: to return the results in a pandas DataFrame.
- early_stopping_rounds: finishes training of the model early if the hold-out metric ("rmse" in our case) does not improve for a given number of rounds.
- seed: for reproducibility of results.

3.1. Diccionario de hiperparámetros (*param*) para pasarselo a un 3-fold cross validation

```
    params = {"objective":"reg:linear",'colsample_bytree': 0.3,'l
earning_rate': 0.1,
                    'max_depth': 5, 'alpha': 10}

    cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold
=3,
                        num_boost_round=50,early_stopping_rounds=
10,metrics="rmse", as_pandas=True, seed=123)
```

3.2. Extract and print the final boosting round metric.

```
print((cv_results["test-rmse-mean"]).tail(1))
```

4. Visualize

Once you train a model using the XGBoost learning API, you can pass it to the plot_tree() function along with the number of trees you want to plot using the num_trees argument.

```
xgb.plot_tree(xg_reg,num_trees=0)
plt.rcParams['figure.figsize'] = [50, 10]
plt.show()
```

Another way to visualize your XGBoost models is to examine the importance of each feature column in the original dataset within the model. XGBoost has a plot_importance() function that allows you to do exactly this.

```
xgb.plot_importance(xg_reg)
plt.rcParams['figure.figsize'] = [5, 5]
plt.show()

F E A T U R E  S E L E C T I O N
```

https://xgboost.readthedocs.io/en/latest/tutorials/index.html (https://xgboost.readthedocs.io/en/latest/tutorials/index.html)

https://www.cienciadedatos.net/documentos/py09_gradient_boosting_python.html

DEALING WITH IMBALANCED DATA

https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18 (https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18)

https://www.kdnuggets.com/2019/05/fix-unbalanced-dataset.html (https://www.kdnuggets.com/2019/05/fix-unbalanced-dataset.html)

- Decision trees frequently perform well on imbalanced data.
- Imbalanced affects to correlation between features.

Oversampling

Oversampling can be a good choice when you don't have a ton of data to work with.

Always split into test and train sets BEFORE trying oversampling techniques! Oversampling before splitting the data can allow the exact same observations to be present in both the test and train sets. This can allow our model to simply memorize specific data points and cause overfitting and poor generalization to the test data.

Undersampling

Undersampling can be a good choice when you have a ton of data -think millions of rows. But a drawback is that we are removing information that may be valuable. This could lead to underfitting and poor generalization to the test set.

Generate synthetic samples (SMOTE)

Again, it's important to generate the new samples only in the training set to ensure our model generalizes well to unseen data.

Conclusión

These are just some of the many possible methods to try when dealing with imbalanced datasets, and not an exhaustive list. Some others methods to consider are collecting more data or choosing different resampling ratios — you don't have to have exactly a 1:1 ratio!

---

https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/ (https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/)

1) Can You Collect More Data?

2) Try Changing Your Performance Metric

- Confusion Matrix: A breakdown of predictions into a table showing correct predictions (the diagonal) and the types of incorrect predictions made (what classes incorrect predictions were assigned).
- Precision: A measure of a classifiers exactness.
- Recall: A measure of a classifiers completeness
- F1 Score (or F-score): A weighted average of precision and recall.

I would also advice you to take a look at the following:

- Kappa (or Cohen's kappa): Classification accuracy normalized by the imbalance of the classes in the data.
- ROC Curves: Like precision and recall, accuracy is divided into sensitivity and specificity and models can be chosen based on the balance thresholds of these values.

5) Try Different Algorithms

6) Try Penalized Models

You can use the same algorithms but give them a different perspective on the problem.

Penalized classification imposes an additional cost on the model for making classification mistakes on the minority class during training. These penalties can bias the model to pay more attention to the minority class.

Often the handling of class penalties or weights are specialized to the learning algorithm. There are penalized versions of algorithms such as penalized-SVM and penalized-LDA.

It is also possible to have generic frameworks for penalized models. For example, Weka has a CostSensitiveClassifier that can wrap any classifier and apply a custom penalty matrix for miss classification.

Using penalization is desirable if you are locked into a specific algorithm and are unable to resample or you're getting poor results. It provides yet another way to "balance" the classes. Setting up the penalty matrix can be complex. You will very likely have to try a variety of penalty schemes and see what works best for your problem.

7) Try a Different Perspective

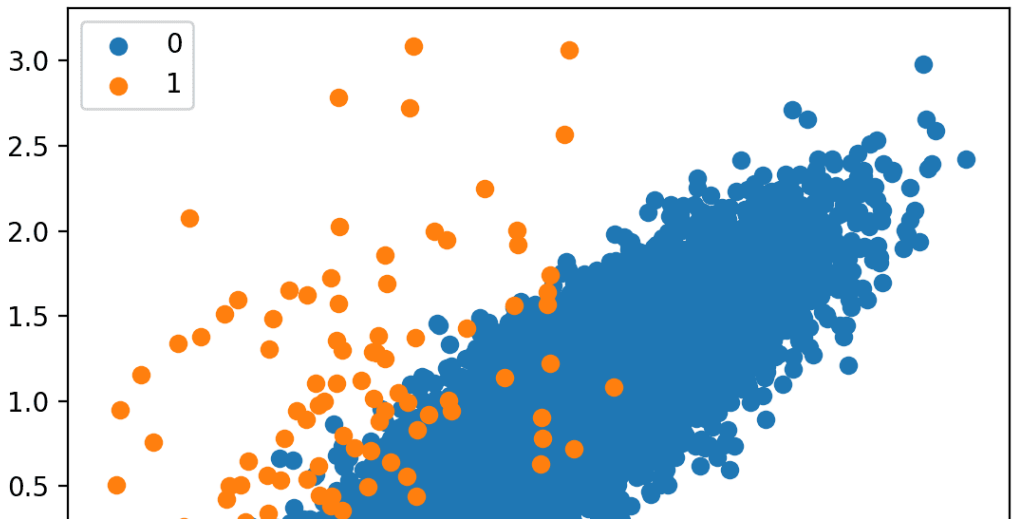
Two you might like to consider are anomaly detection and change detection.

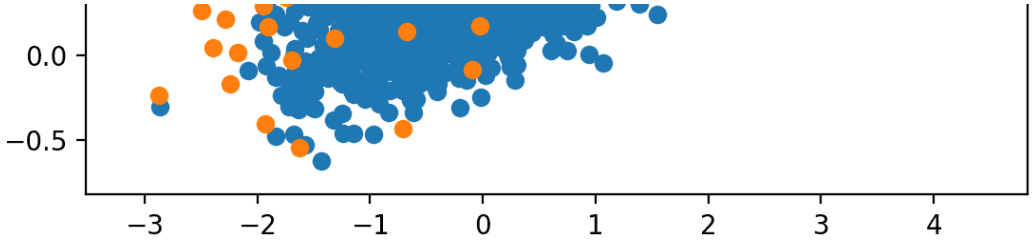
Anomaly detection is the detection of rare events. This might be a machine malfunction indicated through its vibrations or a malicious activity by a program indicated by it's sequence of system calls. The events are rare and when compared to normal operation.

Change detection is similar to anomaly detection except rather than looking for an anomaly it is looking for a change or difference. This might be a change in behavior of a user as observed by usage patterns or bank transactions.

SMOTE

https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/ (https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/)





Importar

```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
```

Oversampling

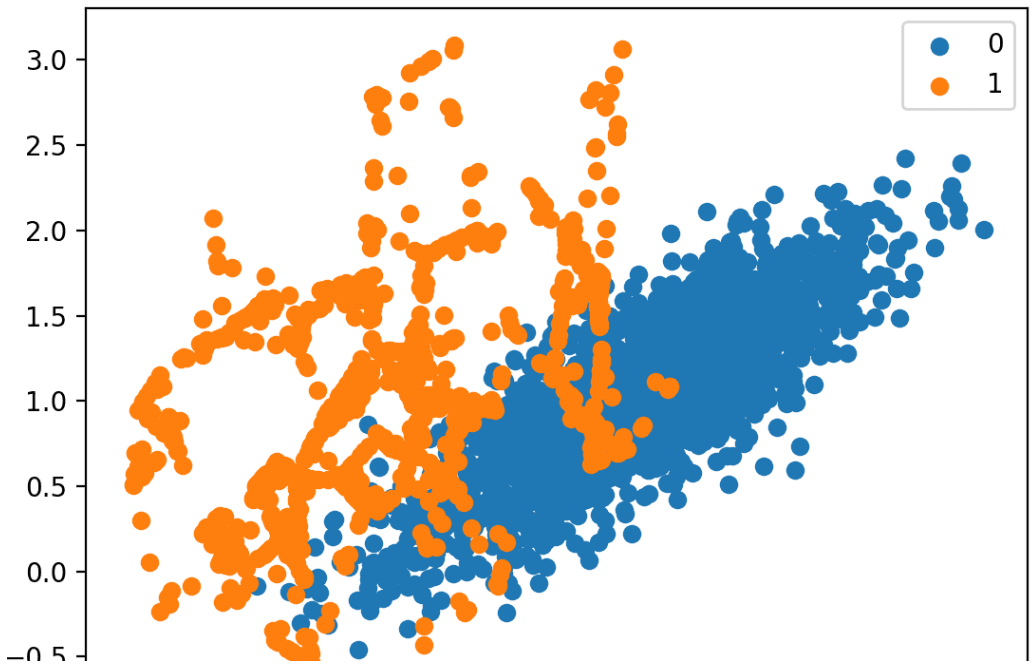
```
oversample = SMOTE()
X, y = oversample.fit_resample(X, y)
```

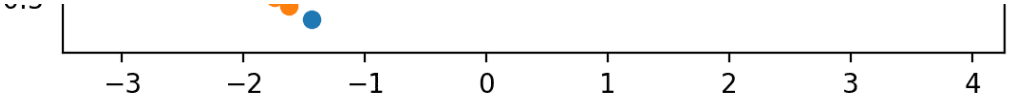
Undersampling and oversampling

```
over = SMOTE(sampling_strategy=0.1)
under = RandomUnderSampler(sampling_strategy=0.5)

steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)

#transform the dataset
X, y = pipeline.fit_resample(X, y)
```





SUMMARIZING CLASS DISTRIBUTION FOR PLOTTING

```
    from collections import Counter
    counter = Counter(y)
    print(counter)

    # scatter plot of examples by class label
    for label, _ in counter.items():
        row_ix = where(y == label)[0]
        pyplot.scatter(X[row_ix, 0], X[row_ix, 1], label=str(labe
l))
    pyplot.legend()
    pyplot.show()
```

SMOTE FOR IMBALANCED CLASSIFICATION

```
    model = DecisionTreeClassifier()
    over = SMOTE(sampling_strategy=0.1)
    under = RandomUnderSampler(sampling_strategy=0.5)
    steps = [('over', over), ('under', under), ('model', model)]
    pipeline = Pipeline(steps=steps)

    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random
_state=1)
    scores = cross_val_score(pipeline, X, y, scoring='roc_auc', c
v=cv, n_jobs=-1)
    print('Mean ROC AUC: %.3f' % mean(scores))
```

You could explore testing different ratios of the minority class and majority class (e.g. changing the sampling_strategy argument) to see if a further lift in performance is possible.

Another area to explore would be to test different values of the k-nearest neighbors selected in the SMOTE procedure when each new synthetic example is created. The default is k=5, although larger or smaller values will influence the types of examples created, and in turn, may impact the performance of the model.

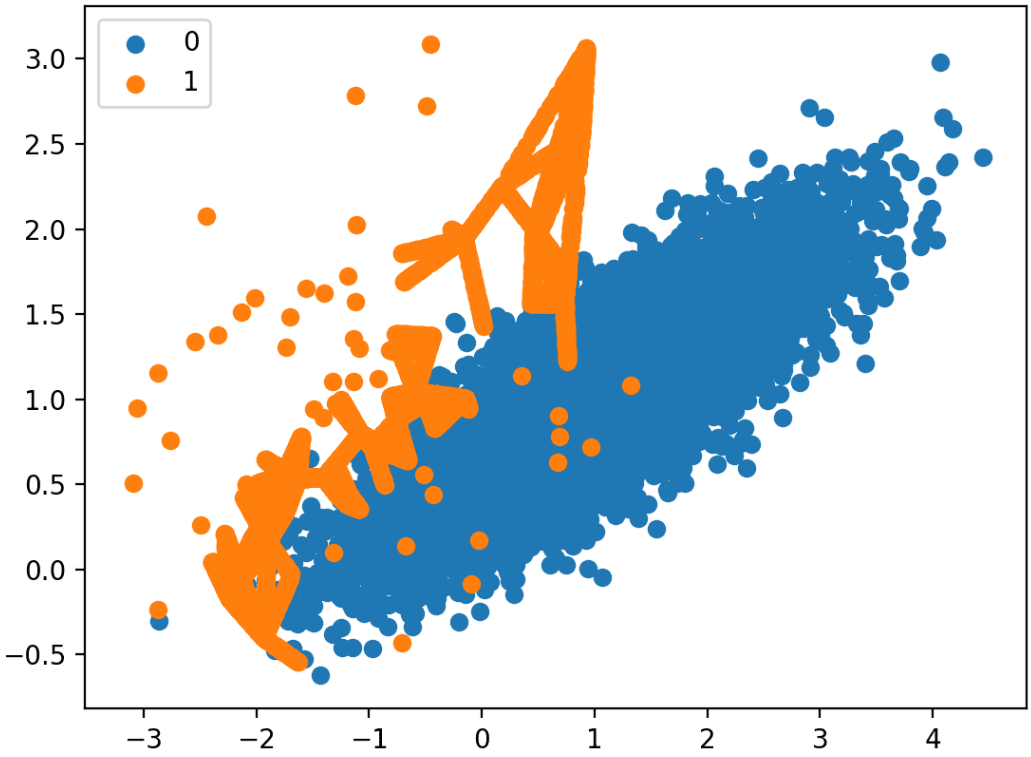
```
    k_values = [1, 2, 3, 4, 5, 6, 7]
    for k in k_values:
        # define pipeline
```

SMOTE With Selective Synthetic Sample Generation

Borderline-SMOTE

Instead of generating new synthetic examples for the minority class blindly, we would expect the Borderline-SMOTE method to only create synthetic examples along the decision boundary between the two classes.

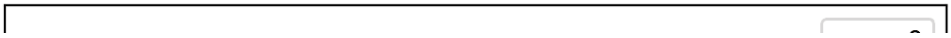
```
from imblearn.over_sampling import BorderlineSMOTE
oversample = BorderlineSMOTE()
X, y = oversample.fit_resample(X, y)
```

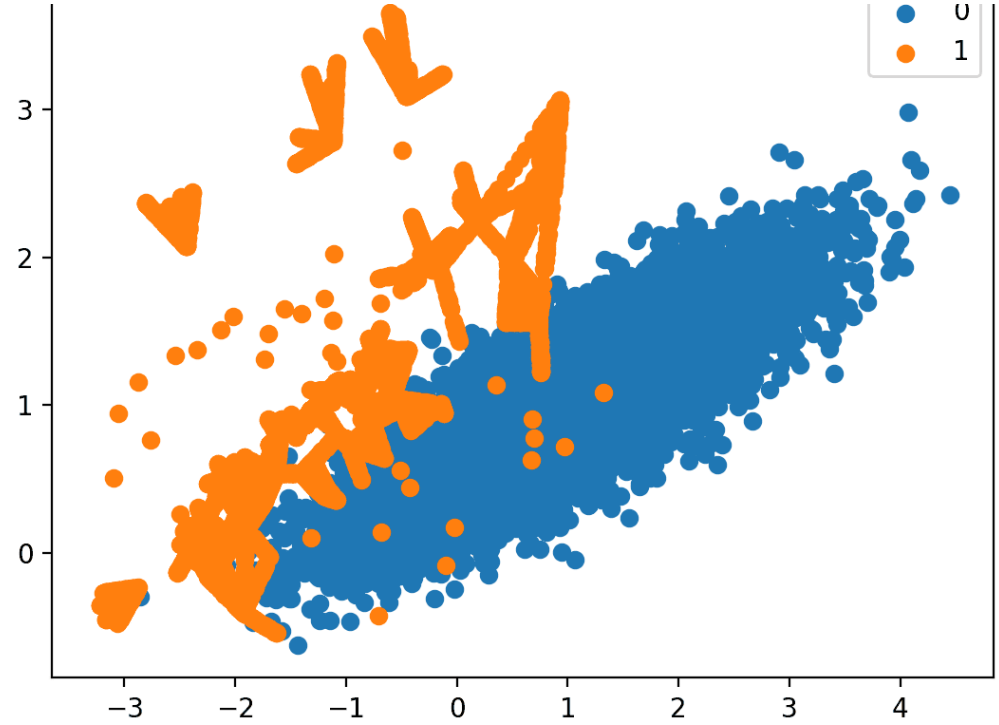


Borderline-SMOTE SVM

An SVM is used to locate the decision boundary defined by the support vectors and examples in the minority class that close to the support vectors become the focus for generating synthetic examples. In addition to using an SVM, the technique attempts to select regions where there are fewer examples of the minority class and tries to extrapolate towards the class boundary.

```
from imblearn.over_sampling import SVMSMOTE
oversample = SVMSMOTE()
X, y = oversample.fit_resample(X, y)
```

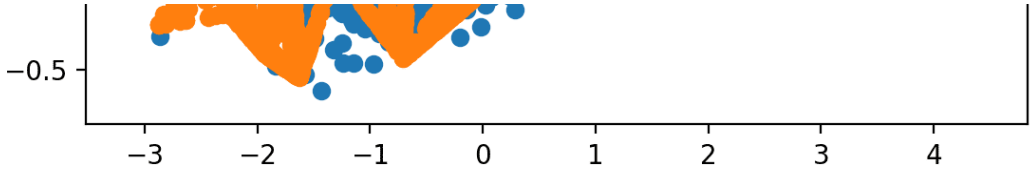




Adaptive Synthetic Sampling (ADASYN)

Another approach involves generating synthetic samples inversely proportional to the density of the examples in the minority class. That is, generate more synthetic examples in regions of the feature space where the density of minority examples is low, and fewer or none where the density is high.

```
from imblearn.over_sampling import ADASYN
oversample = ADASYN()
X, y = oversample.fit_resample(X, y)
```



Unlike Borderline-SMOTE, we can see that the examples that have the most class overlap have the most focus. On problems where these low density examples might be outliers, the ADASYN approach may put too much attention on these areas of the feature space, which may result in

ALGORITMOS NO SUPERVISADOS

https://www.cienciadedatos.net/documentos/py20-clustering-con-python.html (https://www.cienciadedatos.net/documentos/py20-clustering-con-python.html)

https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68 (https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68)

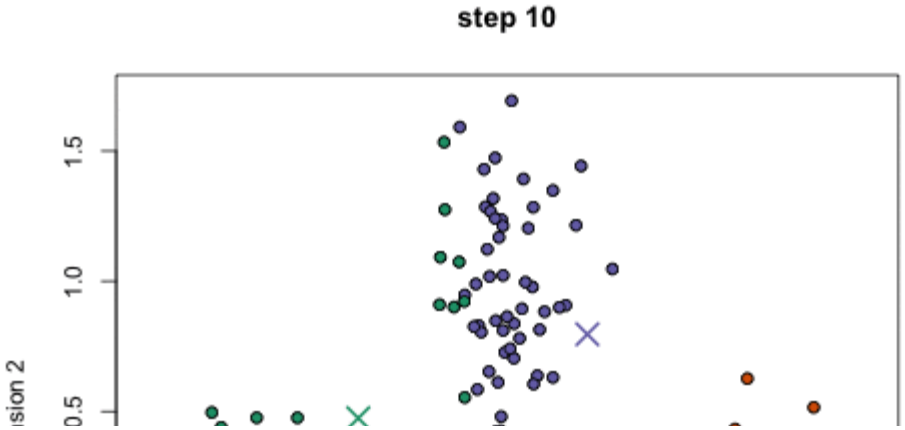
K-MEANS // K-MEDIANS

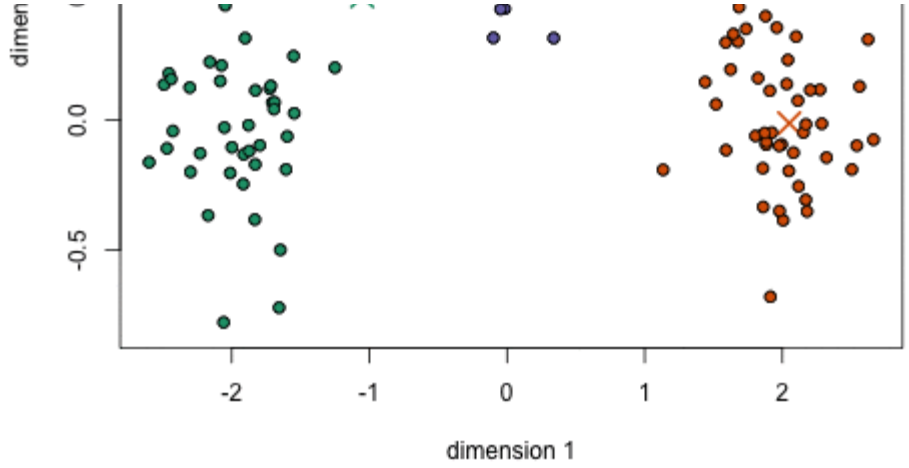
K-Means has the advantage that it's pretty fast, as all we're really doing is computing the distances between points and group centers; very few computations! It thus has a linear complexity O(n).

Disadvantages:

- Firstly, you have to select how many groups/classes there are.
- K-means also starts with a random choice of cluster centers and therefore it may yield different clustering results on different runs of the algorithm. Thus, the results may not be repeatable and lack consistency.

K-Medians is another clustering algorithm related to K-Means, except instead of recomputing the group center points using the mean we use the median vector of the group. This method is **less sensitive to outliers** (because of using the Median) but is **much slower for larger datasets** as sorting is required on each iteration when computing the Median vector.





Code

```
from sklearn.cluster import KMeans
```

Here are the parameters used in this example:

- init controls the initialization technique. The standard version of the k-means algorithm is implemented by setting init to "random". Setting this to "k-means++" employs an advanced trick to speed up convergence, which you'll use later.
- n_clusters sets k for the clustering step. This is the most important parameter for k-means.
- n_init sets the number of initializations to perform. This is important because two runs can converge on different cluster assignments. The default behavior for the scikit-learn algorithm is to perform ten k-means runs and return the results of the one with the lowest SSE.
- max_iter sets the number of maximum iterations for each initialization of the k-means algorithm.

```
kmeans = KMeans(
    init="random",
    n_clusters=3,
    n_init=10,
    max_iter=300,
    random_state=42
)
```

This will perform ten runs of the k-means algorithm on your data with a maximum of 300 iterations per run.

See results

```
# The lowest SSE value
kmeans.inertia_

# Final locations of the centroid
kmeans.cluster_centers_

# The number of iterations required to converge
kmeans.n_iter_
```

Finally, the cluster assignments are stored as a one-dimensional NumPy array in kmeans.labels_. Note that the order of the cluster labels for the first two data objects was flipped. The order was [1, 0] in true_labels but [0, 1] in kmeans.labels_ even though those data objects are still members of their original clusters in kmeans.lables_.

```
kmeans.labels_[:5]
```

Evaluate numer of clusters

- The elbow method

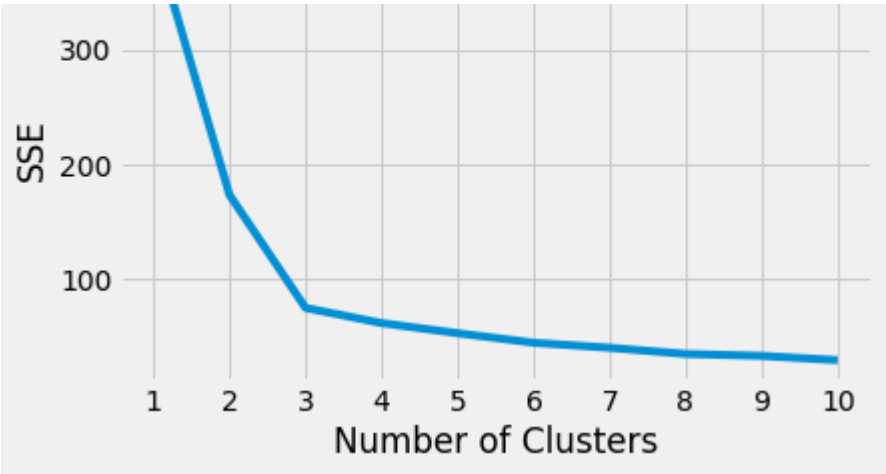
To perform the elbow method, run several k-means, increment k with each iteration, and record the SSE:

```
kmeans_kwargs = {
    "init": "random",
    "n_init": 10,
    "max_iter": 300,
    "random_state": 42,
}

# A list holds the SSE values for each k
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(scaled_features)
    sse.append(kmeans.inertia_)
```

When you plot SSE as a function of the number of clusters, notice that SSE continues to decrease as you increase k. As more centroids are added, the distance from each point to its closest centroid will decrease. There's a sweet spot where the SSE curve starts to bend known as the elbow point. The x-value of this point is thought to be a reasonable trade-off between error and number of clusters. In this example, the elbow is located at x=3:

```
plt.style.use("fivethirtyeight")
 plt.plot(range(1, 11), sse)
 plt.xticks(range(1, 11))
```



use a Python package, kneed, to identify the elbow point programmatically:

```
kl = KneeLocator(
    range(1, 11), sse, curve="convex", direction="decreasing"
)

kl.elbow
```

- The silhouette coefficient

The silhouette coefficient is a measure of cluster cohesion and separation. It quantifies how well a data point fits into its assigned cluster based on two factors:

- How close the data point is to other points in the cluster
- How far away the data point is from points in other clusters

Silhouette coefficient values range between -1 and 1. Larger numbers indicate that samples are closer to their clusters than they are to other clusters.

In the scikit-learn implementation of the silhouette coefficient, the average silhouette coefficient of all the samples is summarized into one score. The **silhouette score()** function needs a minimum of two clusters, or it will raise an exception.

```
# A list holds the silhouette coefficients for each k
silhouette_coefficients = []

# Notice you start at 2 clusters for silhouette coefficient
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(scaled_features)
    score = silhouette_score(scaled_features, kmeans.labels_)
    silhouette_coefficients.append(score)

plt.plot(range(2, 11), silhouette_coefficients)
plt.xticks(range(2, 11))
```

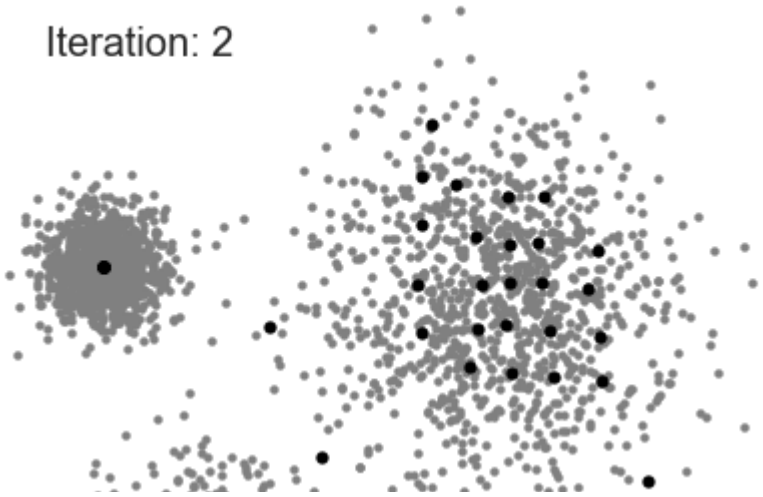
ADVANCED MODEL EVALUATION

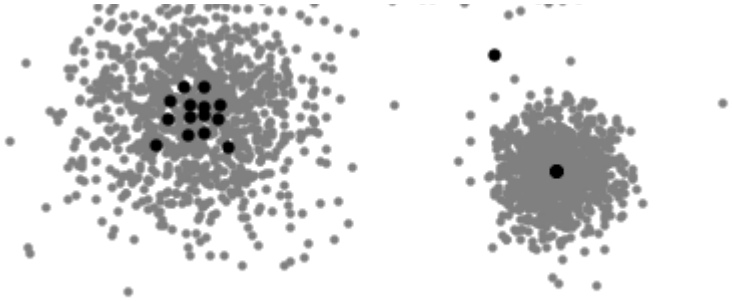
- https://scikit-learn.org/stable/modules/classes.html#clustering-metrics (https://scikit-learn.org/stable/modules/classes.html#clustering-metrics)
- DBSCAN algorithm
- adjusted rand index (ARI)

Mean-Shift Clustering

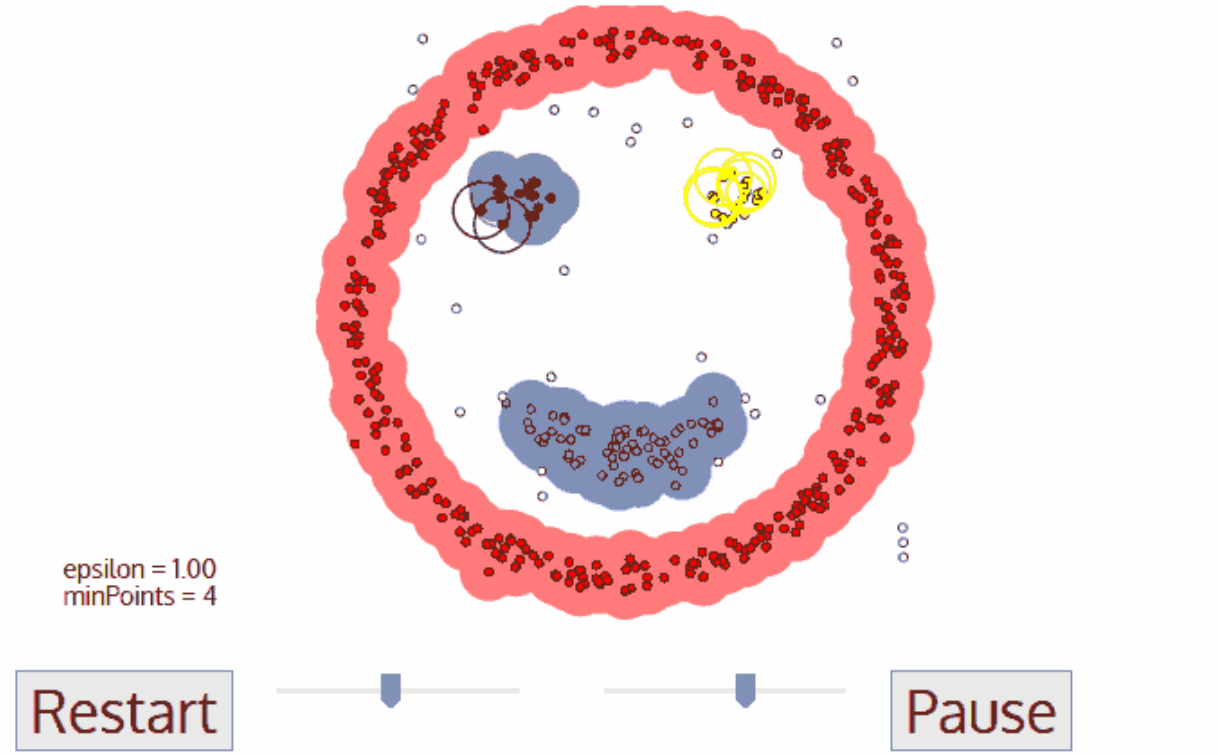
Mean shift clustering is a sliding-window-based algorithm that attempts to find dense areas of data points. It is a centroid-based algorithm meaning that the goal is to locate the center points of each group/class, which works by updating candidates for center points to be the mean of the points within the sliding-window. These candidate windows are then filtered in a post-processing stage to eliminate near-duplicates, forming the final set of center points and their corresponding groups.

In contrast to K-means clustering, there is no need to select the number of clusters as mean-shift automatically discovers this.The fact that the cluster centers converge towards the points of maximum density is also quite desirable as it is quite intuitive to understand and fits well in a naturally data-driven sense. The drawback is that the selection of the window size/radius “r” can be non-trivial.





Density-Based Spatial Clustering of Applications with Noise (DBSCAN)



DBSCAN begins with an arbitrary starting data point that has not been visited. If there are a sufficient number of points (according to **minPoints**) within this neighborhood then the clustering process starts and the current data point becomes the first point in the new cluster. Otherwise, the point will be labeled as noise (later this noisy point might become the part of the cluster). In both cases that point is marked as “visited”.

ADV

- DBSCAN poses some great advantages over other clustering algorithms. Firstly, it does not require a pe-set number of clusters at all. It also identifies outliers as noises, unlike mean-shift which simply throws them into a cluster even if the data point is very different. Additionally, it can find arbitrarily sized and arbitrarily shaped clusters quite well.

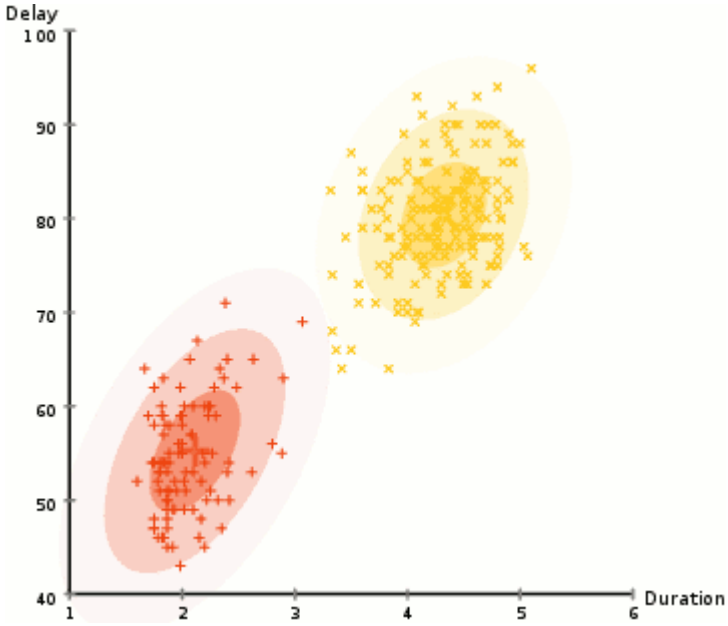
DISADV

- The main drawback of DBSCAN is that it doesn’t perform as well as others when the clusters

are of varying density.
- This drawback also occurs with very high-dimensional data since again the distance threshold ε becomes challenging to estimate.

Expectation–Maximization (EM) Clustering using Gaussian Mixture Models (GMM)

Given these Gaussian distributions for each cluster, compute the probability that each data point belongs to a particular cluster. The closer a point is to the Gaussian's center, the more likely it belongs to that cluster. Based on these probabilities, we compute a new set of parameters for the Gaussian distributions such that we maximize the probabilities of data points within the clusters.



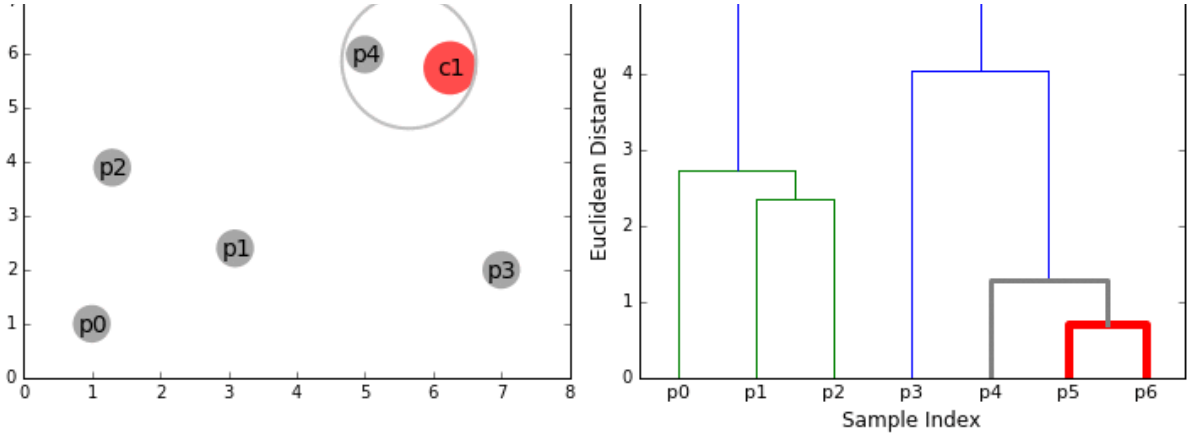
ADV

- Firstly GMMs are a lot more flexible in terms of cluster covariance than K-Means.
- Secondly, since GMMs use probabilities, they can have multiple clusters per data point. So if a data point is in the middle of two overlapping clusters, we can simply define its class by saying it belongs X-percent to class 1 and Y-percent to class 2. I.e GMMs support mixed membership.

Agglomerative Hierarchical Clustering

Hierarchical clustering algorithms fall into 2 categories: top-down or bottom-up.

Bottom-up algorithms treat each data point as a single cluster at the outset and then successively merge (or agglomerate) pairs of clusters until all clusters have been merged into a single cluster that contains all data points. Bottom-up hierarchical clustering is therefore called hierarchical agglomerative clustering or HAC. This hierarchy of clusters is represented as a tree (or dendrogram).



Step 2 is repeated until we reach the root of the tree i.e we only have one cluster which contains all data points. In this way we can select how many clusters we want in the end, simply by choosing when to stop combining the clusters.

ADV

- does not require us to specify the number of clusters and we can even select which number of clusters looks best since we are building a tree.
- the algorithm is not sensitive to the choice of distance metric
- A particularly good use case of hierarchical clustering methods is when the underlying data has a hierarchical structure and you want to recover the hierarchy; other clustering algorithms can't do this. These advantages of hierarchical clustering come at the cost of lower efficiency, as it has a time complexity of $O(n^3)$

---

https://realpython.com/k-means-clustering-python/ (https://realpython.com/k-means-clustering-python/)

Partitional Clustering

Partitional clustering divides data objects into nonoverlapping groups. These techniques require the user to specify the number of clusters, indicated by the variable k. Two examples of partitional clustering algorithms are k-means and k-medoids. These algorithms are both nondeterministic.

Partitional clustering methods have several strengths:

```
They work well when clusters have a spherical shape.
They're scalable with respect to algorithm complexity.
```

They also have several weaknesses:

```
They're not well suited for clusters with complex shapes and diff
erent sizes.
They break down when used with clusters of different densities.
```

Hierarchical Clustering

Hierarchical clustering determines cluster assignments by building a hierarchy. These methods

produce a tree-based hierarchy of points called a dendrogram. hierarchical clustering is deterministic

- Agglomerative clustering is the bottom-up approach. It merges the two points that are the most similar until all points have been merged into a single cluster.
- Divisive clustering is the top-down approach. It starts with all points as one cluster and splits the least similar clusters at each step until only single data points remain.

The strengths of hierarchical clustering methods include the following:

```
They often reveal the finer details about the relationships betwe
en data objects.
They provide an interpretable dendrogram.
```

The weaknesses of hierarchical clustering methods include the following:

```
They're computationally expensive with respect to algorithm compl
exity.
They're sensitive to noise and outliers.
```

Density-Based Clustering

Density-based clustering determines cluster assignments based on the density of data points in a region. Examples of density-based clustering algorithms include Density-Based Spatial Clustering of Applications with Noise, or DBSCAN, and Ordering Points To Identify the Clustering Structure, or OPTICS.

The strengths of density-based clustering methods include the following:

```
They excel at identifying clusters of nonspherical shapes.
They're resistant to outliers.
```

CREACIÓN DE PIPELINES

https://towardsdatascience.com/a-simple-example-of-pipeline-in-machine-learning-with-scikit-learn-e726ffbb6976 (https://towardsdatascience.com/a-simple-example-of-pipeline-in-machine-learning-with-scikit-learn-e726ffbb6976)

https://www.analyticsvidhya.com/blog/2020/01/build-your-first-machine-learning-pipeline-using-scikit-learn/ (https://www.analyticsvidhya.com/blog/2020/01/build-your-first-machine-learning-pipeline-using-scikit-learn/)

https://scikit-learn.org/stable/modules/compose.html#combining-estimators (https://scikit-learn.org/stable/modules/compose.html#combining-estimators)

```
    steps = [('scaler', StandardScaler()), ('SVM', SVC())]from sk
learn.pipeline import Pipeline
    pipeline = Pipeline(steps) # define the pipeline object.
```

- **The final step has to be an estimator in this list of tuples**.

- All estimators in a pipeline, except the last one, must be transformers (i.e. must have a transform method). The last estimator may be any type (transformer, classifier, etc.).

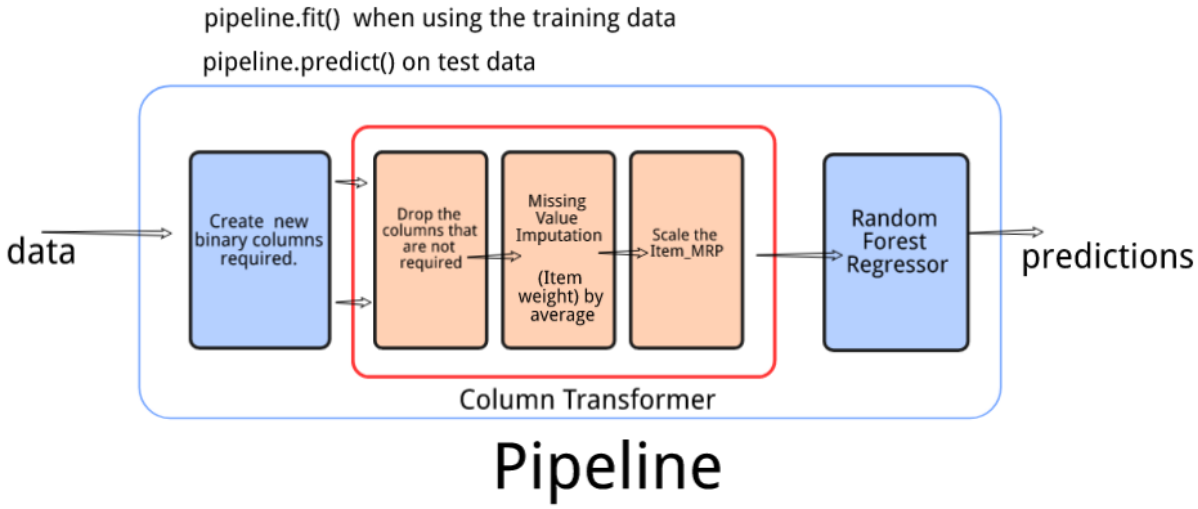
Why Pipelines?

One can bypass this oversimplification by using pipeline. Using pipeline we glue together the StandardScaler() and SVC() and this ensure that during cross validation the StandardScaler is fitted to only the training fold, exactly similar fold used for SVC.fit().

Pipeline design

(MIRAR ANEXO 1)

1. Create the required binary features
2. Perform required data preprocessing and transformations
3. Build a model to predict the sales



Steps

The estimators of a pipeline are stored as a list in the steps attribute, but can be accessed by index or name by indexing (with [idx]) the Pipeline:

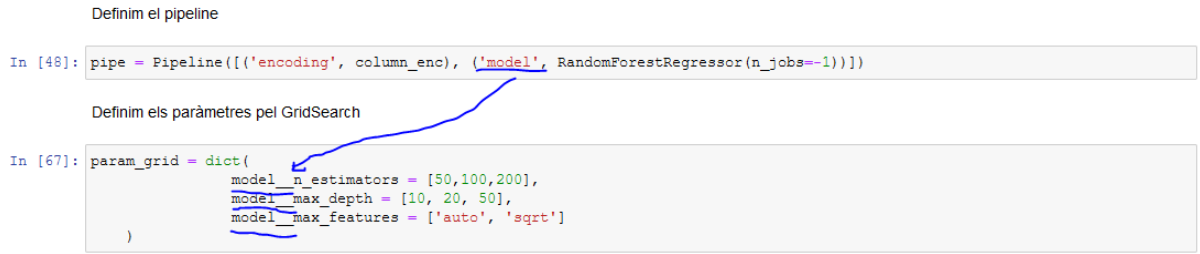
```
pipe.steps[0]
pipe[0]
pipe['reduce_dim']
```

Parameters of the estimators in the pipeline can be accessed using the <estimator>__<parameter> syntax:

```
    pipe.set_params(clf__C=10)
    Pipeline(steps=[('reduce_dim', PCA()), ('clf', SVC(C=10))])

    param_grid = dict(reduce_dim=['passthrough', PCA(5), PCA(1
0)],
                      clf=[SVC(), LogisticRegression()],
                      clf__C=[0.1, 10, 100])
    grid_search = GridSearchCV(pipe, param_grid=param_grid)
```

NOTA: para usar pipeline+gridsearch es necesario introducir el nombre del modelo y dos __ antes de cada parámetro en el diccionario de parámetros.



ColumnTransformer for heterogeneous data

IMPORTANTISSIMO: Column transformer corre todas las transformaciones en paralelo, no es secuencial!!

Many datasets contain features of different types, say text, floats, and dates, where each type of feature requires separate preprocessing or feature extraction steps. Often it is easiest to preprocess data before applying scikit-learn methods, for example using pandas. Processing your data before passing it to scikit-learn might be problematic for one of the following reasons:

1. Incorporating statistics from test data into the preprocessors makes cross-validation scores unreliable (known as data leakage), for example in the case of scalers or imputing missing values.
2. You may want to include the parameters of the preprocessors in a parameter search.

The ColumnTransformer helps performing different transformations for different columns of the data, within a Pipeline that is safe from data leakage and that can be parametrized. ColumnTransformer works on arrays, sparse matrices, and pandas DataFrames.

```
    >>> from sklearn.compose import ColumnTransformer
    >>> from sklearn.feature_extraction.text import CountVectoriz
er
    >>> from sklearn.preprocessing import OneHotEncoder
    >>> column_trans = ColumnTransformer(
    ...     [('city_category', OneHotEncoder(dtype='int'),['city
']),
    ...      ('title_bow', CountVectorizer(), 'title')],
    ...     remainder='drop')

    >>> column_trans.fit(X)
```

==========In the above example, the CountVectorizer expects a 1D array as input and therefore the columns were specified as a string ('title'). However, OneHotEncoder as most of other transformers expects 2D data, therefore in that case you need to specify the column as a list of strings (['city']).==========

EXTRA:

- https://towardsdatascience.com/using-columntransformer-to-combine-data-processing-steps-af383f7d5260 (https://towardsdatascience.com/using-columntransformer-to-combine-data-processing-steps-af383f7d5260)

Developing scikit-learn estimators (Base Estimators)

https://scikit-learn.org/stable/developers/develop.html (https://scikit-learn.org/stable/developers/develop.html)

Hay funciones, clases y estimadores. Los estimadores son básicamente clases cuyos parámetros consisten en fit y en transform (como OneHotEncoder, StandardScaler, etc.). La única diferencia es que con esto te lo haces a tu manera y conveniencia.

```
class OutletTypeEncoder(BaseEstimator):

    def __init__(self):
        pass

    def fit(self, documents, y=None):
        return self

    def transform(self, x_dataset):

        ##Aquí le pongo todo lo que quiero que haga mi estima
tor sobre el x_dataset

        return x_dataset
```

NOTA: La parte de fit no la entiendo muy bien pero por el momento con saber usar el transform

ya me va bien.

EXTRA

- **DIFERENTES TIPOS DE BASE CLASSES COMO BASE ESTIMATORS**: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.base (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.base)
- **Pipelines y custom transformers**: https://towardsdatascience.com/pipelines-custom-transformers-in-scikit-learn-ef792bbb3260 (https://towardsdatascience.com/pipelines-custom-transformers-in-scikit-learn-ef792bbb3260)
- **CUSTOMIZAR PREPROCESAMIENTO Y PIPELINES**: https://gist.github.com/amberjrivera/8c5c145516f5a2e894681e16a8095b5c (https://gist.github.com/amberjrivera/8c5c145516f5a2e894681e16a8095b5c)
- Function transformer: https://kiwidamien.github.io/introducing-the-column-transformer.html

Tuning hyperparameters (GridSearchCV)

- https://scikit-learn.org/stable/modules/grid_search.html#grid-search (https://scikit-learn.org/stable/modules/grid_search.html#grid-search)
- https://scikit-learn.org/0.24/modules/generated/sklearn.model_selection.HalvingGridSearchCV.html (https://scikit-learn.org/0.24/modules/generated/sklearn.model_selection.HalvingGridSearchCV.html)
- (Uso de GridSeach muy interesante) https://github.com/codebasics/py/blob/master/ML/15_gridsearch/15_grid_search.ipynb (https://github.com/codebasics/py/blob/master/ML/15_gridsearch/15_grid_search.ipynb)
- https://www.cienciadedatos.net/documentos/py11-calibrar-modelos-machine-learning.html (https://www.cienciadedatos.net/documentos/py11-calibrar-modelos-machine-learning.html)

TEXT MINING

---> para buscar: classifying documents to information extraction

https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk (https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk)

https://www.cienciadedatos.net/documentos/py25-text-mining-python.html (https://www.cienciadedatos.net/documentos/py25-text-mining-python.html)

Tokenization

Sentence Tokenization

```
from nltk.tokenize import sent_tokenize

tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

Word Tokenization

```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

Frequency Distribution

```
from nltk.probability import FreqDist

fdist = FreqDist(tokenized_word)
print(fdist)

fdist.most_common(2)

fdist.plot(30,cumulative=False)
plt.show()
```

https://stackoverflow.com/questions/27488446/how-do-i-get-word-frequency-in-a-corpus-using-scikit-learn-countvectorizer (https://stackoverflow.com/questions/27488446/how-do-i-get-word-frequency-in-a-corpus-using-scikit-learn-countvectorizer)

cv.vocabulary_ in this instance is a dict, where the keys are the words (features) that you've found and the values are indices, which is why they're 0, 1, 2, 3.

- Counting words with CounVectorizer: https://investigate.ai/text-analysis/counting-words-with-scikit-learns-countvectorizer/ (https://investigate.ai/text-analysis/counting-words-with-scikit-learns-countvectorizer/)

Stopwords

```
from nltk.corpus import stopwords

stop_words=set(stopwords.words("english"))
print(stop_words)


filtered_sent=[]
for w in tokenized_sent:
    if w not in stop_words:
        filtered_sent.append(w)
```

Lexicon Normalization

It reduces derivationally related forms of a word to a common root word.

1. Stemming

Stemming is a process of linguistic normalization, which reduces words to their word root word or chops off the derivational affixes. For example, connection, connected, connecting word reduce to a common word "connect".

2. Lemmatization

Lemmatization reduces words to their base word, which is linguistically correct lemmas. It transforms root word with the use of vocabulary and morphological analysis. Lemmatization is usually more sophisticated than stemming. Stemmer works on an individual word without knowledge of the context. For example, The word "better" has "good" as its lemma. This thing will miss by stemming because it requires a dictionary look-up.

```
from nltk.stem.wordnet import WordNetLemmatizer
lem = WordNetLemmatizer()

from nltk.stem.porter import PorterStemmer
stem = PorterStemmer()

word = "flying"
print("Lemmatized Word:",lem.lemmatize(word,"v"))
print("Stemmed Word:",stem.stem(word))

Lemmatized Word: fly
Stemmed Word: fli
```

POS Tagging

The primary target of Part-of-Speech(POS) tagging is to identify the grammatical group of a given word. Whether it is a NOUN, PRONOUN, ADJECTIVE, VERB, ADVERBS, etc. based on the context. POS Tagging looks for relationships within the sentence and assigns a corresponding tag to the word.

```
nltk.pos_tag(tokens)
```

Sentiment Analysis

There are mainly two approaches for performing sentiment analysis.

- Lexicon-based: count number of positive and negative words in given text and the larger count will be the sentiment of text.
- Machine learning based approach: Develop a classification model, which is trained using the pre-labeled dataset of positive, negative, and neutral.

```
    from sklearn.feature_extraction.text import CountVectorizer
    from nltk.tokenize import RegexpTokenizer

    #tokenizer to remove unwanted elements from out data like sy
  mbols and numbers
    token = RegexpTokenizer(r'[a-zA-Z0-9]+')
    cv = CountVectorizer(lowercase=True,stop_words='english',ngr
  am_range = (1,1),tokenizer = token.tokenize)
    text_counts= cv.fit_transform(data['Phrase'])
```

==========

```
  from sklearn.naive_bayes import MultinomialNB
  #Import scikit-learn metrics module for accuracy calculation
  from sklearn import metrics
  # Model Generation Using Multinomial Naive Bayes
  clf = MultinomialNB().fit(X_train, y_train)
  predicted= clf.predict(X_test)
  print("MultinomialNB Accuracy:",metrics.accuracy_score(y_tes
t, predicted))
```

Feature Generation using TF-IDF

In Term Frequency(TF), you just count the number of words occurred in each document. The main issue with this Term Frequency is that it will give more weight to longer documents. Term frequency is basically the output of the BoW model.

IDF(Inverse Document Frequency) measures the amount of information a given word provides across the document. IDF is the logarithmically scaled inverse ratio of the number of documents that contain the word and the total number of documents.

TF-IDF(Term Frequency-Inverse Document Frequency) normalizes the document term matrix. It is

In [ ]:

In [ ]:

MASTERCLASS MACHINE LEARNING

- Paquete lazypredict: https://pypi.org/project/lazypredict/ (https://pypi.org/project/lazypredict/)
- Agrupament K-Means, Random Forest, Suport Vector Machine (SVM)
- Buscar hiperparámetres
- Quant més paràmetres, més lents.
- Solucions per lentitud: Mostra dels atributs, google collab., menys paràmetres.

In [ ]:

In [ ]:

EXTRA

DEEP LEARNING

https://d2l.ai/chapter_preface/index.html (https://d2l.ai/chapter_preface/index.html)

Cosis

- 'verbose' se usa para ir viendo qué está haciendo la máquina
- DICTIONARY UNPACKING OPERATOR: https://realpython.com/iterate-through-dictionary-python/#using-the-dictionary-unpacking-operator (https://realpython.com/iterate-through-dictionary-python/#using-the-dictionary-unpacking-operator)

ANEXO

1. Pipeline ejemplo

```
# importing required libraries
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
import category_encoders as ce
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline

# read the training data set
data = pd.read_csv('dataset/train_kOBLwZA.csv')

# top rows of the data
data.head()

# seperate the independent and target variables
train_x = data.drop(columns=['Item_Outlet_Sales'])
train_y = data['Item_Outlet_Sales']
```

==========

```
    # import the BaseEstimator
    from sklearn.base import BaseEstimator

    # define the class OutletTypeEncoder
    # This will be our custom transformer that will create 3 new
binary columns
    # custom transformer must have methods fit and transform
    class OutletTypeEncoder(BaseEstimator):

        def __init__(self):
            pass

        def fit(self, documents, y=None):
            return self

        def transform(self, x_dataset):
            x_dataset['outlet_grocery_store'] = (x_dataset['Outle
t_Type'] == 'Grocery Store')*1
            x_dataset['outlet_supermarket_3'] = (x_dataset['Outle
t_Type'] == 'Supermarket Type3')*1
            x_dataset['outlet_identifier_OUT027'] = (x_dataset['O
utlet_Identifier'] == 'OUT027')*1

            return x_dataset
```

==========

```
# pre-processsing step
# Drop the columns -
# Impute the missing values in column Item_Weight by mean
# Scale the data in the column Item_MRP
pre_process = ColumnTransformer(remainder='passthrough',
                                transformers=[('drop_columns', 'd
rop', ['Item_Identifier',

'Outlet_Identifier',

'Item_Fat_Content',

'Item_Type',

'Outlet_Identifier',

'Outlet_Size',

'Outlet_Location_Type',

'Outlet_Type'

]),
                                                ('impute_item_weigh
t', SimpleImputer(strategy='mean'), ['Item_Weight']),
                                                ('scale_data', Stan
dardScaler(), ['Item_MRP'])])
```