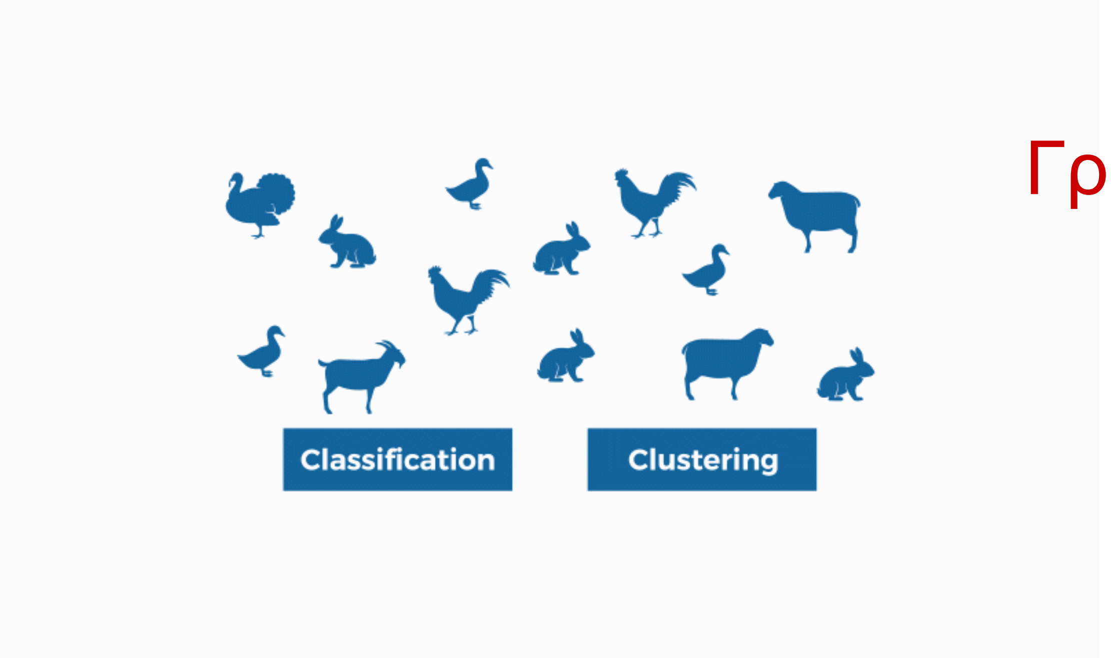
EPL448: Data Mining on the Web – Lab 9

University of Cyprus
Department of
Computer Science

Παύλος Αντωνίου

Γραφείο: B109, ΘΕΕ01



Classification

- Supervised Machine Learning (ML) process of predicting the class or category of data based on a set of predefined classes
- Classification algorithms are trained on datasets containing labeled data
- Labeled data is data observations that have already been classified
  - For example, in a dataset of emails, each email might be labeled as "spam" or "not spam“.
  - These labels (target variable) then provide a clear guide for a classification machine learning algorithm to learn from.

Types of Classification

- Binary Classification
  - process of classification in which input data observations are being classified into one of two discrete classes
  - For example, a medical process which classifies patients into those that have a specific disease versus those that do not (e.g. COVID positive vs COVID negative)
- Multi-class Classification
  - Process of classification in which input data observations are being classified into one of three or more classes
  - For example, medical profiling that classifies patients into those with kidney, liver, lung, or stomach infection symptoms

Classification algorithms

- Popular algorithms used for both binary and multi-class classification:
  - Logistic Regression
  - Support Vector Classifier (SVC)
  - Multinomial Logistic Regression
  - k-Nearest Neighbors (kNN)
  - Decision Tree Classifier
  - Gaussian Naive Bayes Classifier
  - Stochastic Gradient Descent Classifier
  - Linear Discriminant Analysis
  - Ensemble algorithms (Random Forest, Gradient Boosting, XGBoost)

Is rescaling/unskewing needed?

- Feature scaling is recommended prior training classification algorithms that use the notion of (Euclidean) distance between data points to determine their similarity (whether they belong to the same class or category) such as:
  - k-Nearest Neighbors (kNN)
    - Normalization (MinMax scaler) is usually more effective than standardization for KNN because all features are mapped to the same range of values (e.g. between 0 and 1)
  - Support Vector Classifier (SVC)
    - Standardization (Standard or Robust scaler) is usually more effective when the rbf kernel is used in SVC because rbf assumes that features are centered around zero
  - Stochastic Gradient Descent Classifier (SGDClassifier)
    - Standardization (Standard or Robust scaler) benefits SGDClassifier because it allows faster, more stable convergence, preventing biases caused by feature scales

Is rescaling/unskewing needed?

- Unskewing techniques (e.g. BoxCox, Yeo-Johnson, Sqrt, Log) are generally recommended on highly skewed features. In a related study[1], the use of BoxCox transformation has been shown to increase the accuracy of various classifiers (Linear Classifier, KNN, SVC, Bayesian)
- Rescaling/unskewing of target variable (that includes class/category values) does not make any sense in classification problems
- There is no way to know in advance if feature rescaling or unskewing will provide better prediction results. You can always start by fitting your model to (a) raw, (b) normalized, (c) standardized and (d) unskewed data and then comparing the prediction performance of each model

[1] L. Blum, M. Elgendi, C. Menon "Impact of Box-Cox Transformation on Machine-Learning Algorithms", Frontiers in Artificial Intelligence 5:877569, April 2022

Classification model evaluation metrics

- True Positive (TP): When you predict an observation belongs to a class and it actually does belong to that class
  - Correctly (true) predicted positive class
  - A passenger who is classified as COVID positive and is actually positive

- True Negative (TN): When you predict an observation does not belong to a class and it actually does not belong to that class
  - Correctly (true) predicted negative class
  - A passenger who is classified as not COVID positive (negative) and is actually not COVID positive (negative)

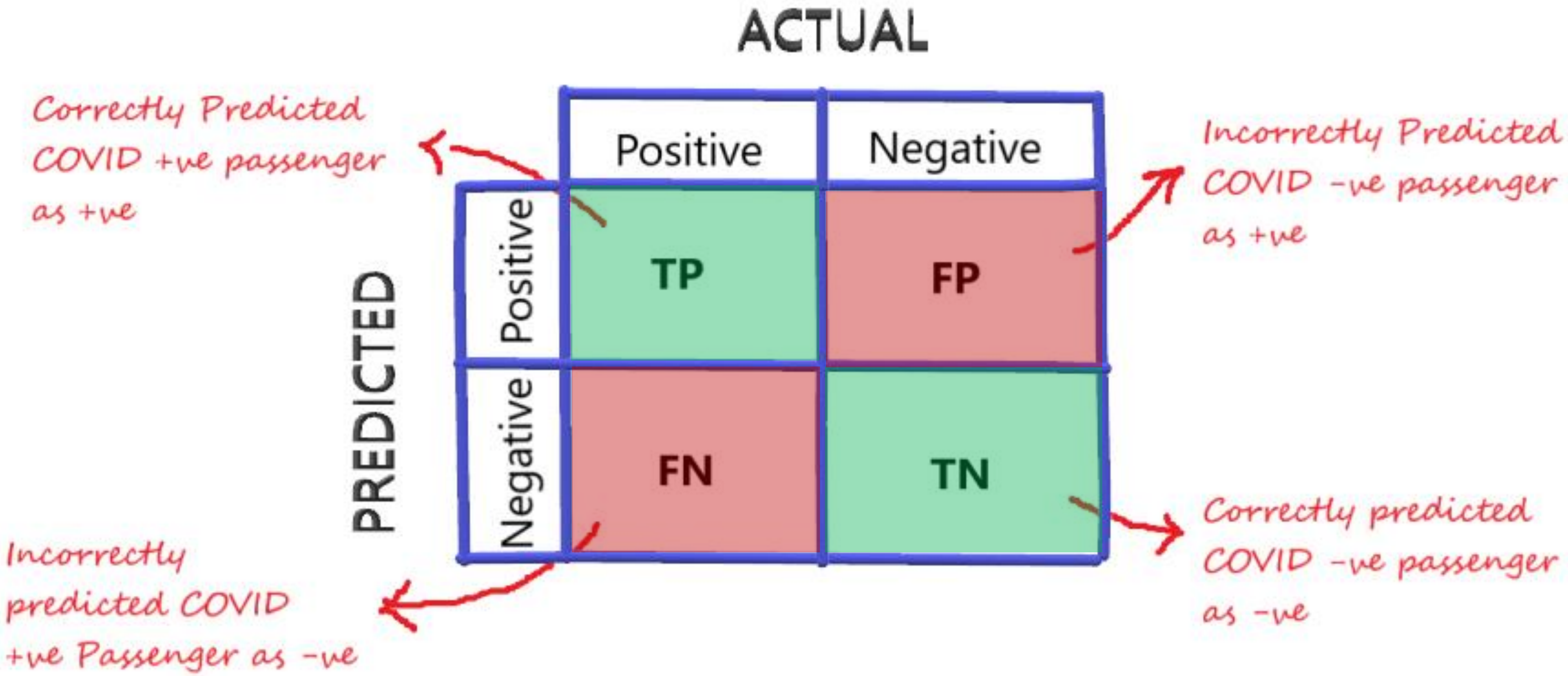
Classification model evaluation metrics

- False Positive (FP): When you predict an observation belongs to a class and it actually does not belong to that class
  - Incorrectly (false) predicted positive class
  - A passenger who is classified as COVID positive and is actually not COVID positive (negative)

- False Negative (FN): When you predict an observation does not belong to a class and it actually does belong to that class
  - Incorrectly (false) predicted negative class
  - A passenger who is classified as not COVID positive (negative) and is actually COVID positive

Confusion matrix

- A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier")



		ACTUAL	
		Positive	Negative
PREDICTED	Positive	TP	FP
	Negative	FN	TN

Correctly Predicted COVID +ve passenger as +ve

Incorrectly Predicted COVID -ve passenger as +ve

Incorrectly predicted COVID +ve Passenger as -ve

Correctly predicted COVID -ve passenger as -ve

Accuracy

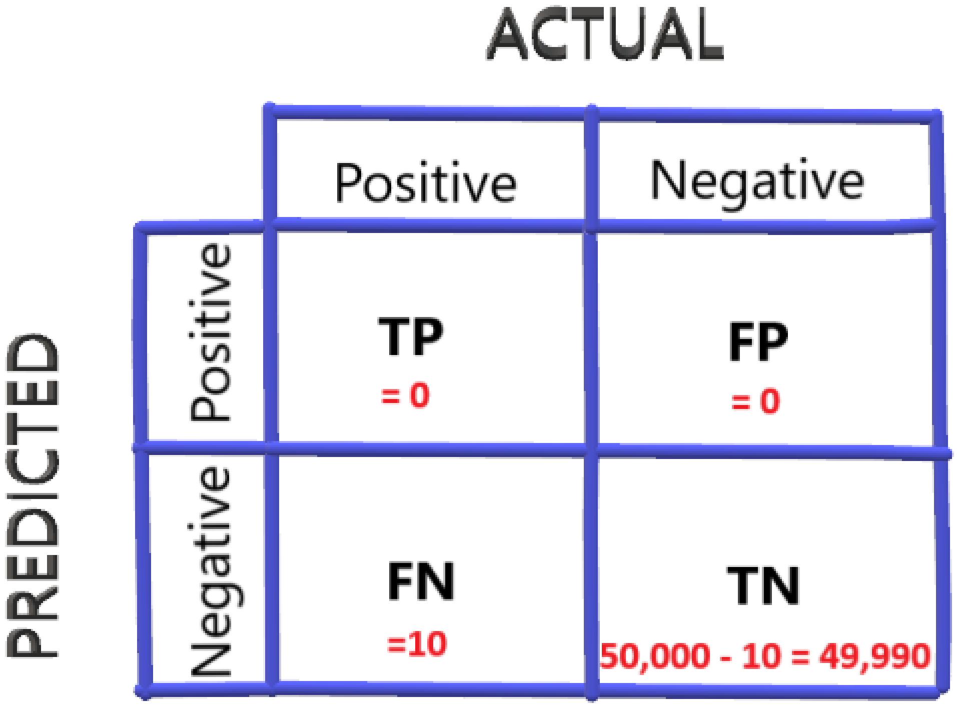
- Accuracy is one metric which gives the fraction of predictions our model got right

- $Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} = \frac{TP+TN}{TP+FP+TN+FN}$

- Ranges from 0 to 1

Is accuracy a good metric?

- Now, let’s consider 50,000 passengers travel per day on an average. Out of which, 10 are actually COVID positive.
- One of the easiest ways to increase the accuracy is to classify every passenger as COVID negative. So, our confusion matrix looks like:



		ACTUAL	
		Positive	Negative
PREDICTED	Positive	TP = 0	FP = 0
	Negative	FN =10	TN 50,000 - 10 = 49,990

$$Accuracy = \frac{49,990}{50,000} = 0.9998 \text{ or } 99.98\%$$

- We achieve more accuracy than we have ever seen in any model, but this does not solve our purpose which is:
- We need to identify COVID positive passengers!

Is accuracy a good metric?

- Not labeling 10 of actually positive passengers entering the country will result in increasing the spread in the community

- Accuracy in this context is a terrible measure because its easy to get extremely good accuracy but that’s not what we are interested in

- But is accuracy always a poor measure? When the **data is balanced, accuracy is a good measure** of evaluating a model. On the other hand if **data is imbalanced** (as in our case), then **accuracy is not a correct measure** of evaluation
  - What is data imbalance: number of samples between classes is uneven

Recall (Sensitivity or True Positive rate)

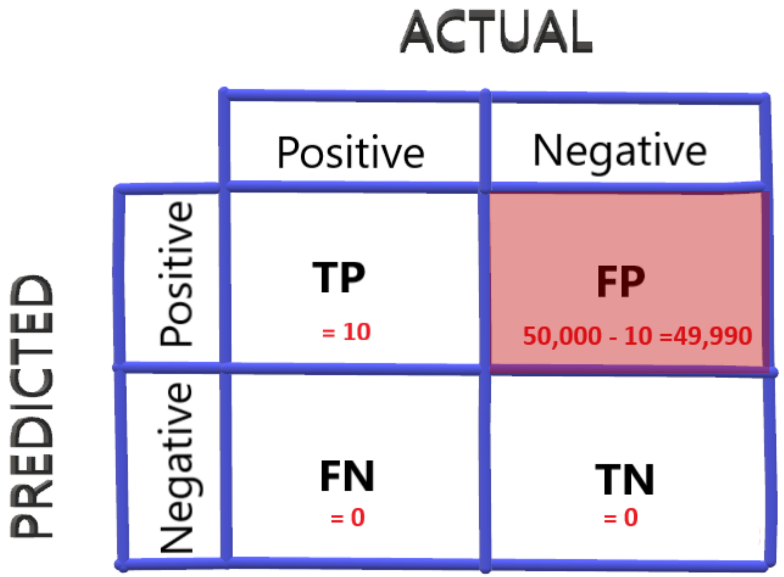
- Recall gives the fraction you correctly identified as positive out of all actual positives – a measure of a classifier's completeness
  - how "sensitive" the classifier is to detecting positive cases
- $Recall = \frac{Number\ of\ correct\ positives}{All\ positives} = \frac{TP}{TP+FN}$
  - Out of all positive passengers what fraction you identified correctly
  - Going back to our previous strategy of labeling every passenger as COVID negative that will give recall of zero: Recall = 0/10 = 0
  - So, in this context, Recall is a good measure. It says that the terrible strategy of identifying every passenger as COVID negative leads to zero recall
  - We want to maximize the recall → 1
  - Is Recall alone good enough to evaluate the performance of a classification model?

Recall

- To answer the previous question, consider another scenario of labeling every passenger as COVID positive

- The confusion matrix will look like:
- $Recall = \frac{TP}{TP+FN} = \frac{10}{10+0} = 1$

- So, recall independently may not a good measure



ACTUAL

PREDICTED		Positive	Negative
	Positive	TP = 10	FP 50,000 - 10 =49,990
	Negative	FN = 0	TN = 0

Precision

- Fraction of correctly identified as positive out of all predicted as positives – a measure of a classifier's exactness
  - Refers to a model's ability to correctly interpret positive observations

- $Precision = \frac{Number\ of\ correct\ positives}{All\ predicted\ positives} = \frac{TP}{TP+FP}$

- Considering our second bad strategy of labeling every passenger as positive, the precision would be :

- $Precision = \frac{TP}{TP+FP} = \frac{10}{10+49990} = 0.0002$

Recall vs Precision

- While this bad strategy has a good recall value of 1 but it has a terrible precision value of 0.0002
- This clarifies that recall alone is not a good measure, we need to consider precision value as well
- Considering another case of labeling only one passenger (correctly) as COVID positive whereas the rest as COVID negative. The confusion matrix in this case will be:

$$Precision = \frac{TP}{TP + FP} = \frac{1}{1 + 0} = 1$$

$$Recall = \frac{TP}{TP + FN} = \frac{1}{1 + 9} = 0.1$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} = \frac{49991}{50000} = 0.99984$$

		ACTUAL	
		Positive	Negative
PREDICTED	Positive	**TP** = 1	**FP** = 0
	Negative	**FN** = 9	**TN** 50,000 - 9 = 49,991

Recall vs Precision

- In some cases, we want to maximize either recall or precision at the cost of others
  - As in this case of labeling passengers, we really want to get the predictions right for COVID positive passengers because it is really expensive to not predict the passenger right as allowing COVID positive person to proceed will result in increasing the spread. So, we are more interested in recall here.
- Unfortunately, sometimes it's difficult to have it both ways: often, increasing precision reduces recall and vice versa. This is called precision/recall tradeoff.

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

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

High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

F-1 score

- Often convenient to combine precision and recall into a single metric
- F1 score is the harmonic mean of the model's precision and recall
- $F1\ score = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$
- Why harmonic mean and not simple average?
  - Not sensitive to extremely large values, unlike simple averages
  - Example: a model with a precision of 1, and recall of 0 gives a simple average as 0.5 and an F-1 score of 0
  - If one of the parameters is low, the second one no longer matters in the F-1 score.
  - F-1 score favors classifiers that have similar precision and recall
  - F-1 score is a better measure to use if you are seeking a balance between Precision and Recall

F-1 score

- Previous formula can be only used in a binary classification problem
- In a multi-class classification problem, we obtain **one F1-score per class** (instead of a single overall F1-score)
  - instead of having multiple per-class F1-scores, it would be better to average them to obtain a single number to describe overall performance
    - Macro average: mean of all the per-class F1 scores. This method treats all classes equally regardless of the number of samples in each class. Not recommended for unbalanced datasets.
    - Weighted average: weighted mean of all the per-class F1 scores. Each class F1-score is multiplied by the percentage of samples belonging to this class (e.g. majority class is given higher weight). **Recommended for unbalanced multi-class datasets**.
    - Micro average: computes a global average F1 score by counting the sums of the True Positives (TP), False Negatives (computes the proportion of correctly classified observations FN), and False Positives (FP) for all classes collectively. In other words, it out of all observations which is the same as measuring the accuracy.

```
from sklearn.metrics import f1_score
f1_score(y_true, y_pred, average='macro')
```

or `weighted` or `micro`

F-beta score

- The F-beta score calculation follows the same form as the F-1 score, however it also allows you to decide how to weight the balance between precision and recall using the beta parameter

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

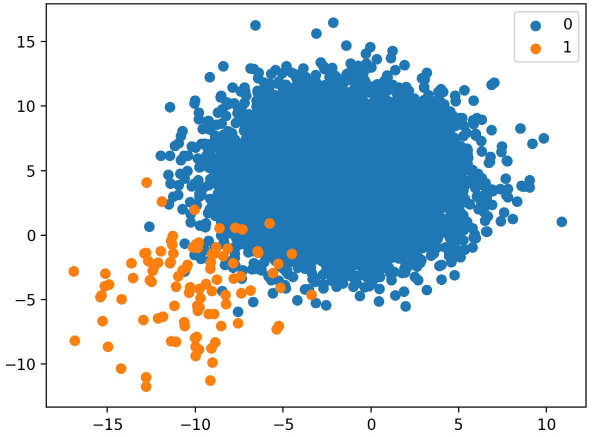
- $$Fbeta\ score = \frac{(1+beta^2) * Precision * Recall}{beta^2 * Precision + Recall}$$

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

  - When beta=1, the F-beta score is equivalent to the F-1 Score
  - When beta=0.5, this score is the F-0.5 score which raises the importance of precision and lowers the importance of recall (goal: minimize False Positives)
  - When beta=2, this score is the F-2 score which raises the importance of recall and lowers the importance of precision (goal: minimize False Negatives)

Balanced vs unbalanced target variable

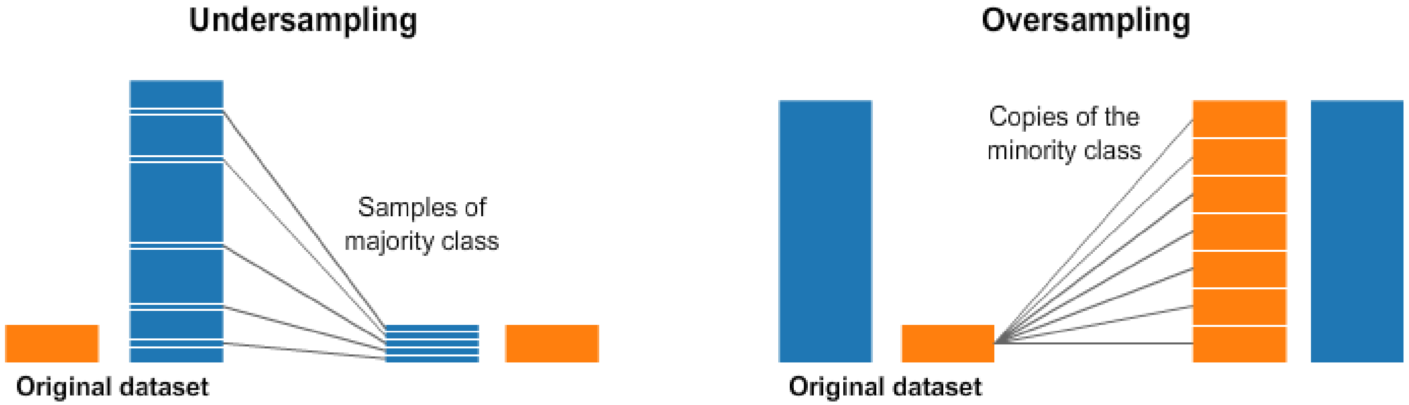
- Data imbalance: typical problem for real-world datasets:
  - number of samples between classes is uneven
    - When size of the majority class gets more than twice the size of the minority class, dataset begins to be considered unbalanced



- Machine learning model tends to be better at predicting the class with more samples (majority class) than the other with fewer samples (minority classes)
  - The greater this imbalance, the higher the bias of the model towards the majority class
- Class imbalance is affected by both the **absolute size of the minority class** and the **imbalance ratio**. Even with enough data, extreme class imbalance can still cause the model to perform poorly on the minority class, necessitating additional strategies to address it.

Balanced vs unbalanced target variable

- Methods for balancing data are available – see here



Synthetic Minority Oversampling Technique (SMOTE)

- However, the problem of (artificially) balanced data can be worse than the unbalanced case – see Appendix (model affected by rare points)
- Effective metrics for unbalanced datasets:
  - Precision, Recall, F-1 score (with weighted average)

Balanced vs unbalanced target variable

- Rule of thumb: you can always track performance of unbalanced classification with Precision/Recall/F-1 score metrics first and then decide whether you need to proceed towards balancing or not
  - If the F1 score is acceptably high and both precision and recall are reasonable, you may decide that balancing isn't necessary potentially saving time and computational resources
    - **High Scores**: Typically **0.8 or above** (80%+) → strong model performance
    - **Moderate Scores**: Between **0.6 and 0.8** (60%-80%) → model is doing reasonably well but may still miss a portion of minority class instances (try applying balancing techniques)
  - if the metrics are low, they indicate that the model struggles with minority class predictions, signaling a need to consider techniques such as undersampling or oversampling (e.g., SMOTE) to help balance performance
    - **Low Scores**: Below **0.6** (less than 60%) → model struggles to recognize the minority class or is heavily biased toward the majority class

Stratified Cross Validation

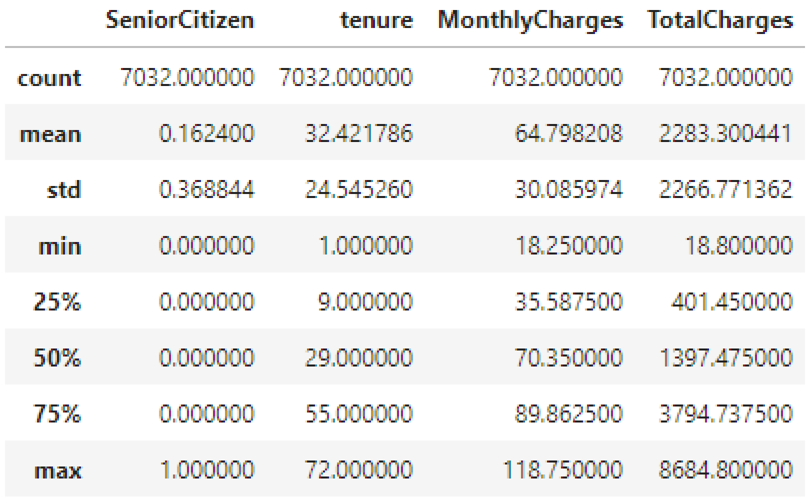
- Let’s assume you are doing a multiclass classification and have an imbalanced dataset that has 5 different classes. The randomized train-test split used in Kfold CV totally disregards the percentage of samples for each class in the resulting splits
- What happens in this scenario is that you end up with train and a test sets with totally different data distributions for each class?
  - A model trained on a vastly different data distribution than the test set, will perform poorly at evaluation step
- The solution to this problem is called stratification
  - StratifiedKFold is a variation of KFold which preserves the percentage of samples for each class in train and test sets
- In GridSearchCV, StratifiedKFold is automatically used when estimator is classifier, and y is either binary or multiclass

Example: Telco Dataset

- A fictional telco company that provided home phone and Internet services to 7043 customers in California in Q3
- Dataset available here
  - 11 rows have missing values => removed
- Each row represents a customer with 21 features
  - Both categorical and numerical
- Target value: Churn – customers decision whether to leave (Yes/No)
- Binary classification problem
  - Predict whether a customer will leave or stay at the end of the quarter

Explore dataset

`df.describe()`

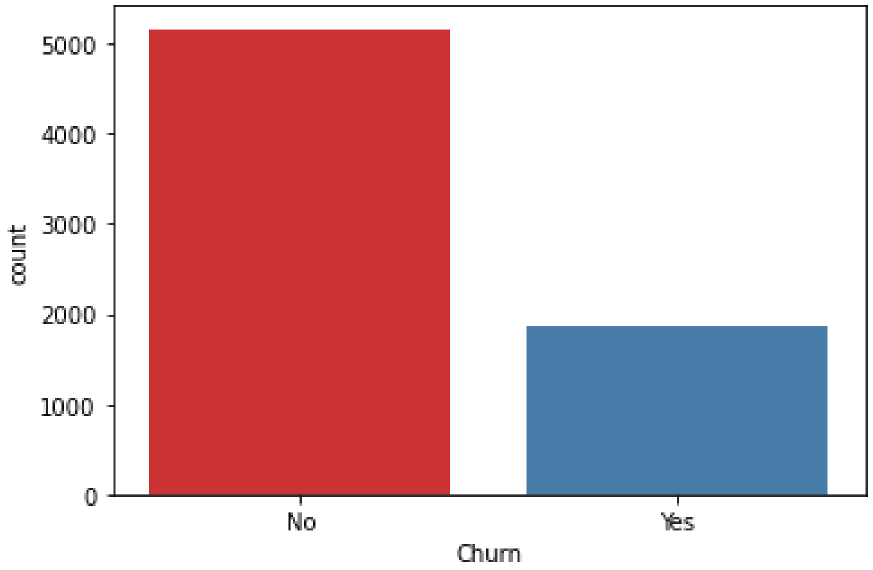


	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	7032.000000	7032.000000	7032.000000	7032.000000
mean	0.162400	32.421786	64.798208	2283.300441
std	0.368844	24.545260	30.085974	2266.771362
min	0.000000	1.000000	18.250000	18.800000
25%	0.000000	9.000000	35.587500	401.450000
50%	0.000000	29.000000	70.350000	1397.475000
75%	0.000000	55.000000	89.862500	3794.737500
max	1.000000	72.000000	118.750000	8684.800000

- 4 Numerical Features
  - SeniorCitizen
    - customer is 65 or older: 1 (Yes), 0 (No)
  - Tenure
    - months that the customer has been with the company
  - MonthlyCharges
    - customer’s current total monthly charge for all their services
  - TotalCharges
    - Tenure*MonthlyCharges
- 16 Categorical Features (most of them are Yes/No, other are categorical)
  - Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, streaming TV and streaming movies
  - Customer account information – id, contract, payment method, paperless billing
  - Demographic info about customers – gender, and if they have partners and dependents
- Target Variable: Churn

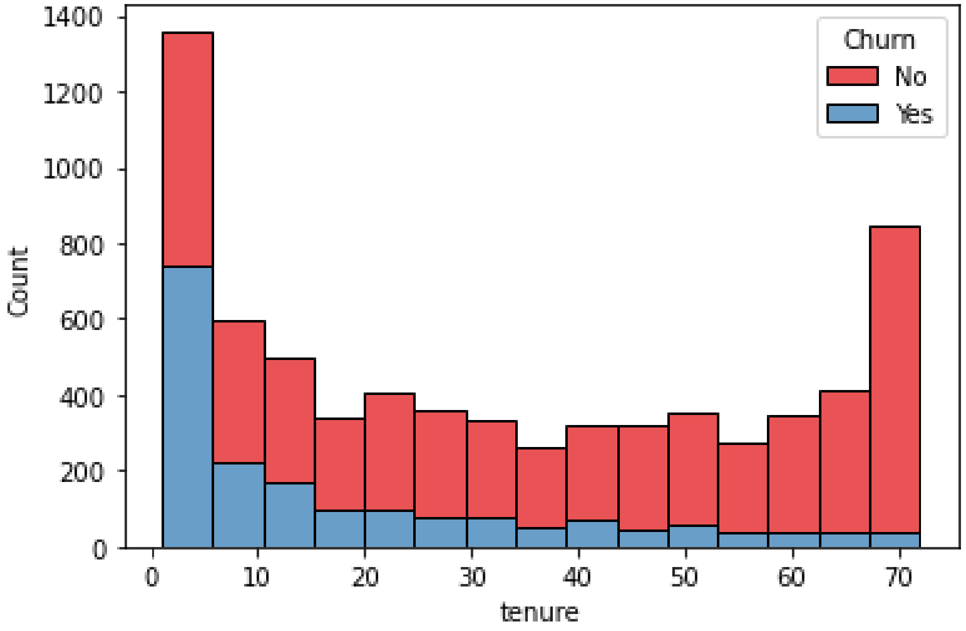
Observations: Unbalanced

- Dataset (target variable) is imbalanced
  - Churn “No" is almost 3 times as “Yes"
  - Accuracy is not the right model evaluation metric and it seems we need to consider Precision, Recall and F-1/F-beta Score



Observations: Tenure vs Churn

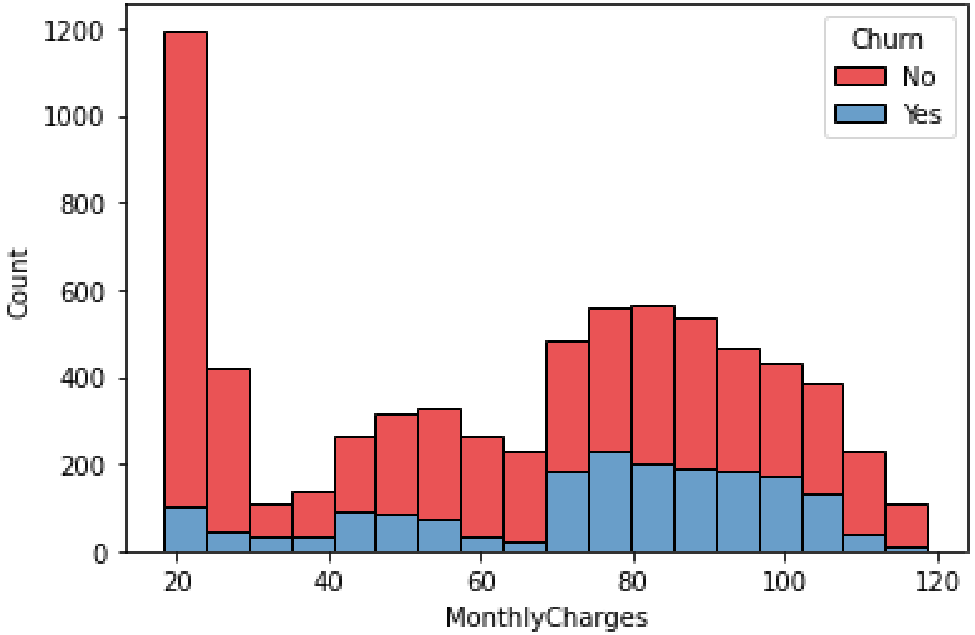
- Customer who left the Telco are mostly customers within 1st month (600+) and churn steady declines over time
- If customer can be retained between 10-20 months, there are high chances, customer will stay very long
- Customer at 72-month tenure, mostly stayed (Churn=0)
- Tenure seems to be a significant feature since its values are significantly related to the customer churn rate (high variance in churn rate for different values of tenure)



(number of months with the company)

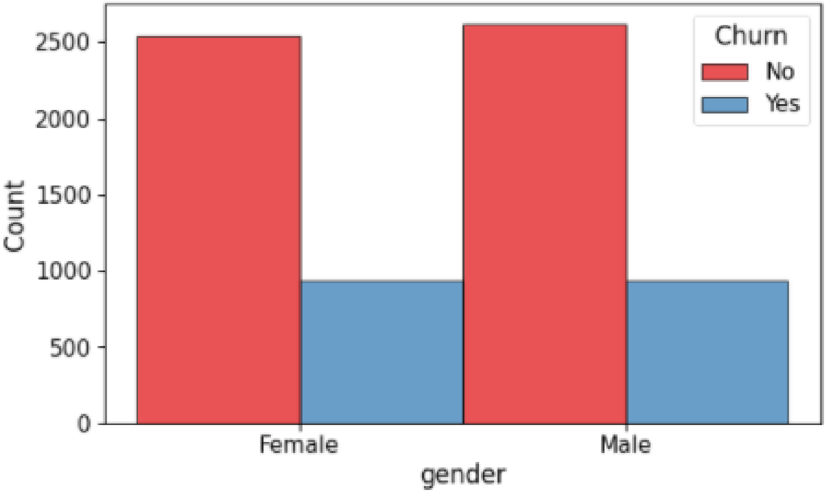
Observations: MonthlyCharges vs Churn

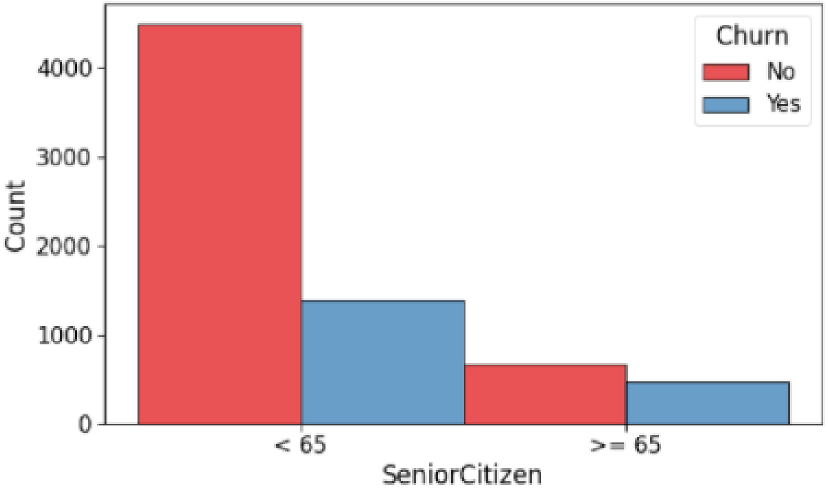
- Majority of customers pay low small monthly charges ($18 – 20) and tend to be loyal
- Customer leaving are mostly in the band of $75-100 who have opted for multiple services
- MonthlyCharges seems to be a significant feature that can used to predict which customers are expected to leave



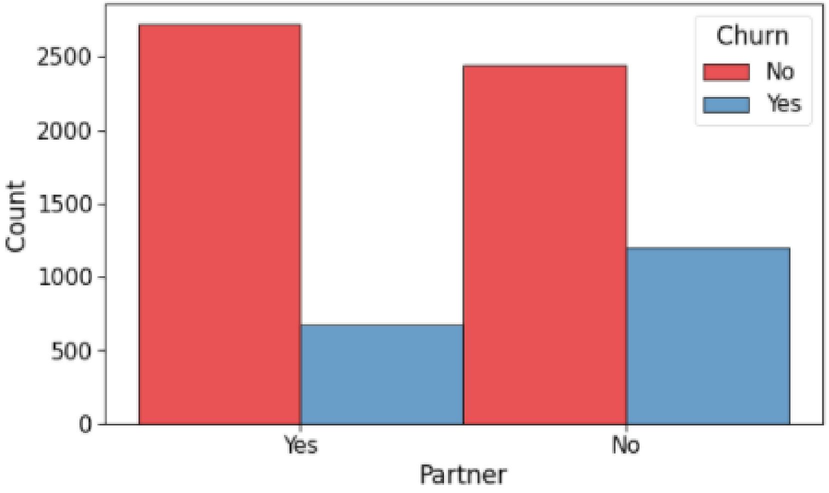
Observations

- gender: Difficult to determine churn using this field. Counts are almost same in either category – not significant feature
- SeniorCitizen: Almost 50% of senior customers tend to leave
  - Since the share of senior customers is 16% of the total amount of customers, this indicator requires further research with additional data
- Partner: Customers with partner have lower chance of leaving





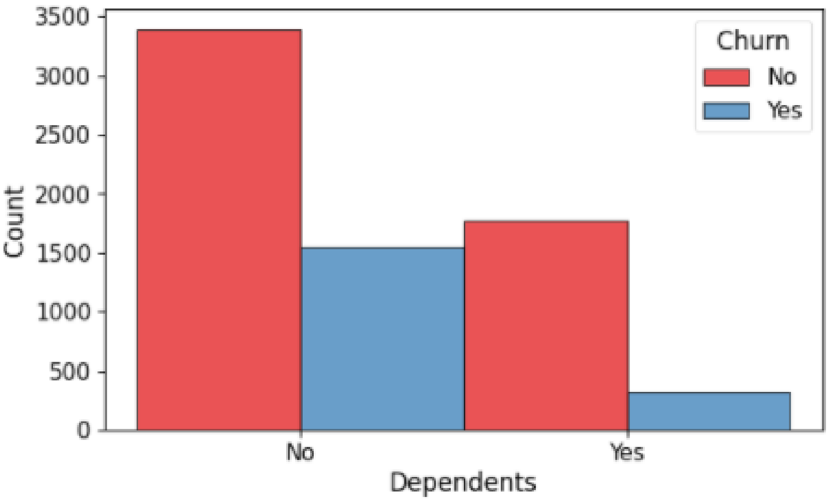
Significant feature



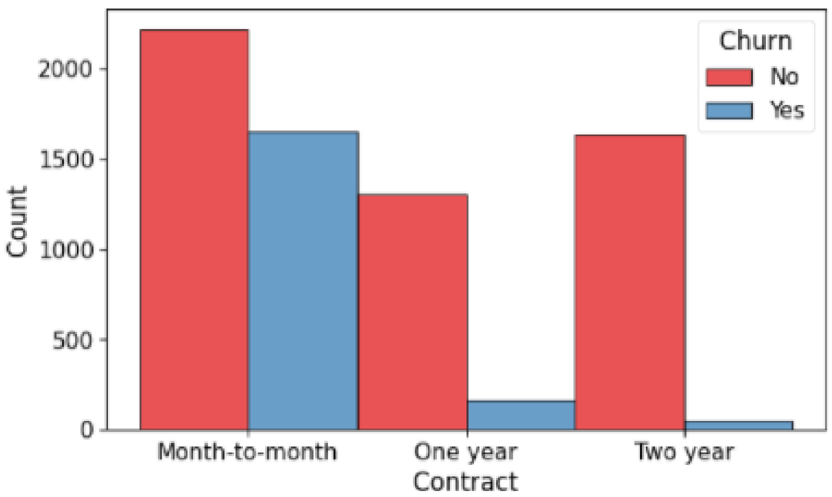
Significant feature

Observations

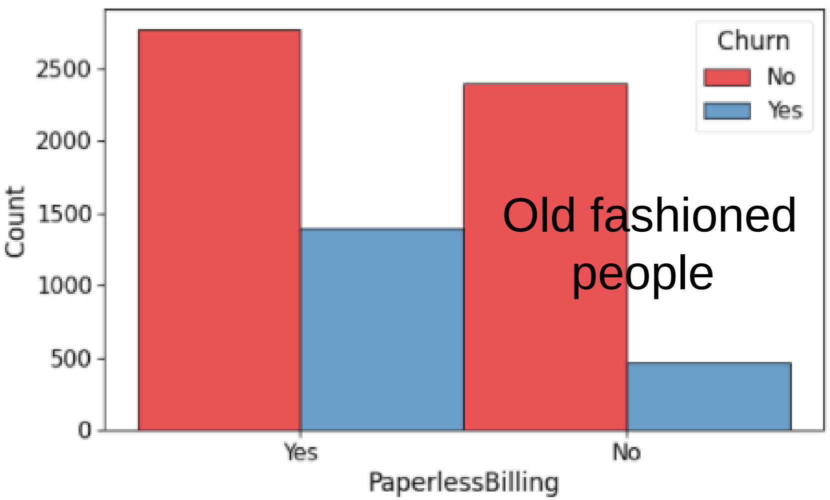
- Dependents: Customer with dependents have lower chance of leaving
- Contract: Month to Month customers have likely higher chances to leave; Old customers are more likely to stay
- PaperlessBilling: Customers with paperless billing have higher chances of leaving compared to more customers preferring traditional paper billing



Significant feature



Significant feature

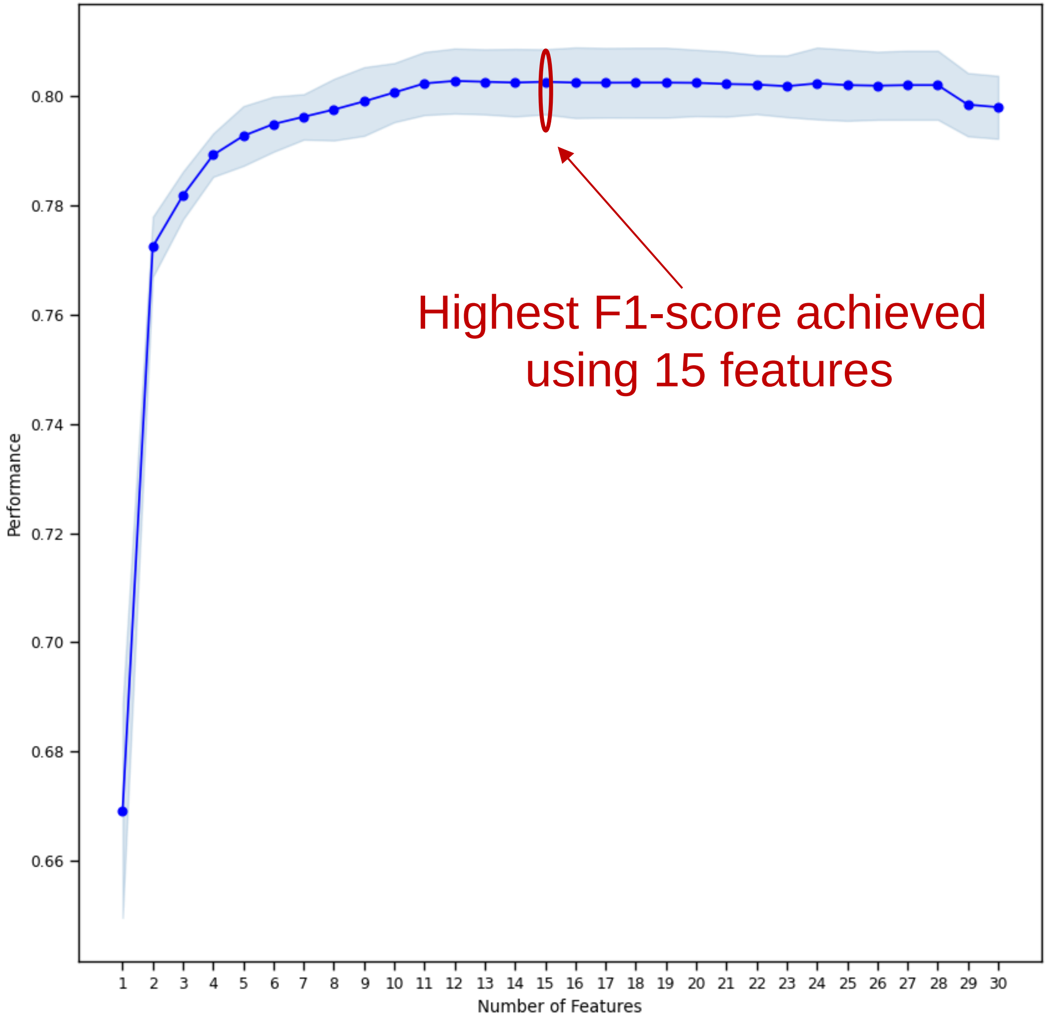


Old fashioned people

Significant feature

Feature Selection (SFS technique)

- Categorical data are converted to numerical using one hot encoding
  - End up with 30 numerical features
- Sequential Forward Selection (SFS) technique used for feature selection (eliminination)
  - keep 15 most important features that maximize the weighted F-1 score (weighted average)

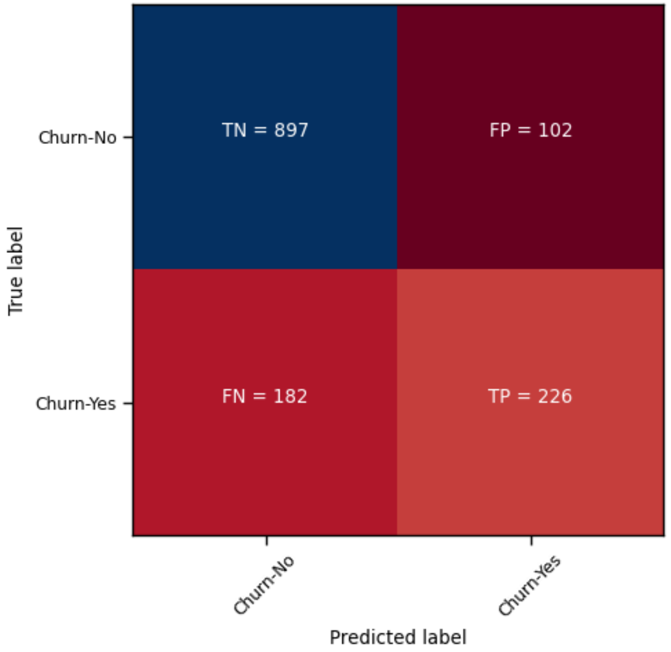


Highest F1-score achieved using 15 features

Initial Evaluation

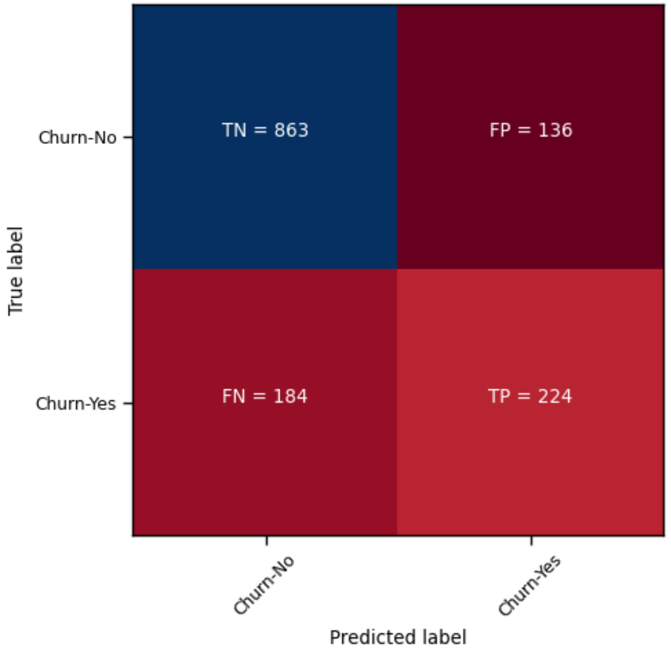
- Run a large set of classifiers using default hyper-parameters values (no GridSearchCV) with 10-fold CV to have an initial feeling of the performance
  - LogisticRegression, KNeighborsClassifier*, SGDClassifier*,GaussianNB, SVC*, DecisionTreeClassifier, RandomForestClassifier, AdaBoostClassifier
  - Consider confusion matrix, Precision, Recall, F-1 score (weighted average)

Logistic Regression Results



Recall=0.503
Precision=0.639
Weighted F1=0.784

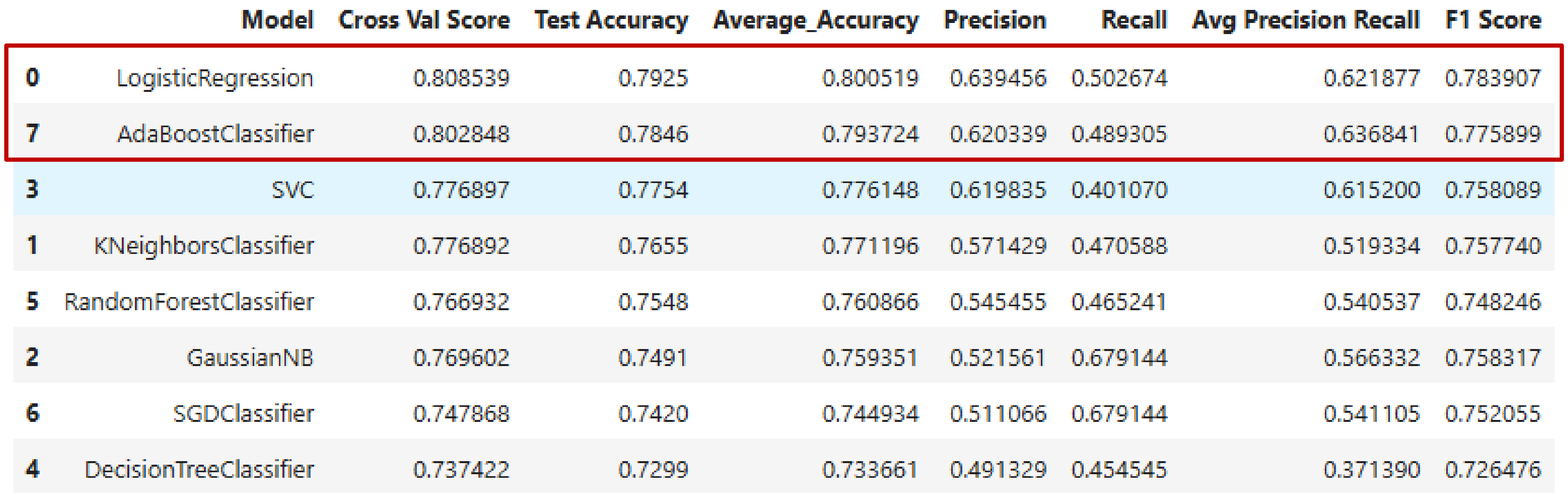
K Nearest Neighbors Results



Recall=0.471
Precision=0.571
Weighted F1=0.758

(*) Feature re-scaling is recommended prior training. It was not applied in these experiments though.

Initial Evaluation: results



	Model	Cross Val Score	Test Accuracy	Average_Accuracy	Precision	Recall	Avg Precision Recall	F1 Score
0	LogisticRegression	0.808539	0.7925	0.800519	0.639456	0.502674	0.621877	0.783907
7	AdaBoostClassifier	0.802848	0.7846	0.793724	0.620339	0.489305	0.636841	0.775899
3	SVC	0.776897	0.7754	0.776148	0.619835	0.401070	0.615200	0.758089
1	KNeighborsClassifier	0.776892	0.7655	0.771196	0.571429	0.470588	0.519334	0.757740
5	RandomForestClassifier	0.766932	0.7548	0.760866	0.545455	0.465241	0.540537	0.748246
2	GaussianNB	0.769602	0.7491	0.759351	0.521561	0.679144	0.566332	0.758317
6	SGDClassifier	0.747868	0.7420	0.744934	0.511066	0.679144	0.541105	0.752055
4	DecisionTreeClassifier	0.737422	0.7299	0.733661	0.491329	0.454545	0.371390	0.726476

- Logistic Regression and Adaboost classifier model look promising achieving highest Average Accuracy and Weighted F1 score
- Let's try to improve both models by selecting the best combination of hyper-parameter values (use of GridSearchCV)

Best model selection

- GridSearchCV on a grid hyper parameters
  - Logistic Regression and Ada Boost Classifier do not require feature scaling
- Logistic Regression Classifier

```
Final accuracy score on the testing data: 0.7925
Final F-score on the testing data: 0.7839
LogisticRegression(C=10, max_iter=10000, solver='newton-cg’)
```

- AdaBoost Classifier

```
Final accuracy score on the testing data: 0.7846
Final F1-score on the testing data: 0.7748
AdaBoostClassifier(algorithm='SAMME', n_estimators=500)
```

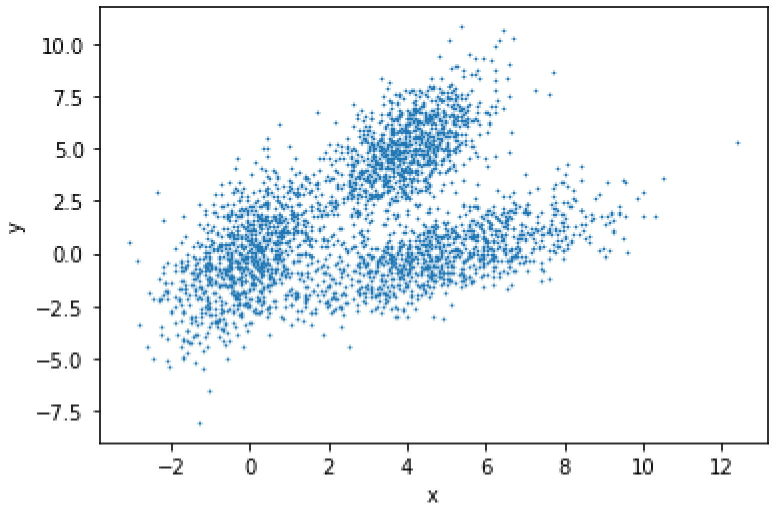
- Source code for all classification experimentations can be found here

Best practices for approaching a data science project

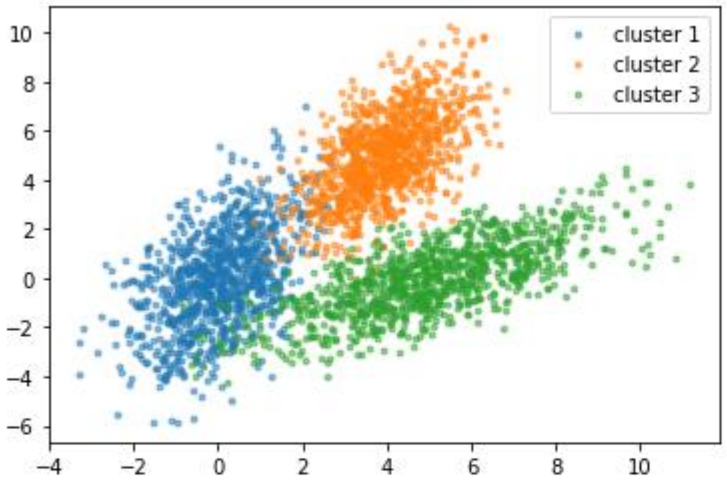
- Jupyter notebook acting as a comprehensive roadmap for approaching a data science project, covering all major steps - from exploratory data analysis and data preprocessing to hyperparameter tuning - to create the most effective predictive model:
  - Exploratory Data Analysis (EDA)
  - dataset splitting early on to avoid data leakage
  - missing values imputation
  - encoding (categorical data -> numerical data)
  - feature scaling, encoding, unskewing
  - feature extraction
  - automated hyperparameter tuning using GridSearchCV with pipelines and parameter grids
- Available here

Clustering

- Unsupervised Machine Learning process (no target variable) of dividing the dataset into groups consisting of similar data points
- Each group is called a cluster and contains data points with high similarity and low similarity with data points in other clusters



Samples in two-dimensional (2 features) space
BEFORE clustering

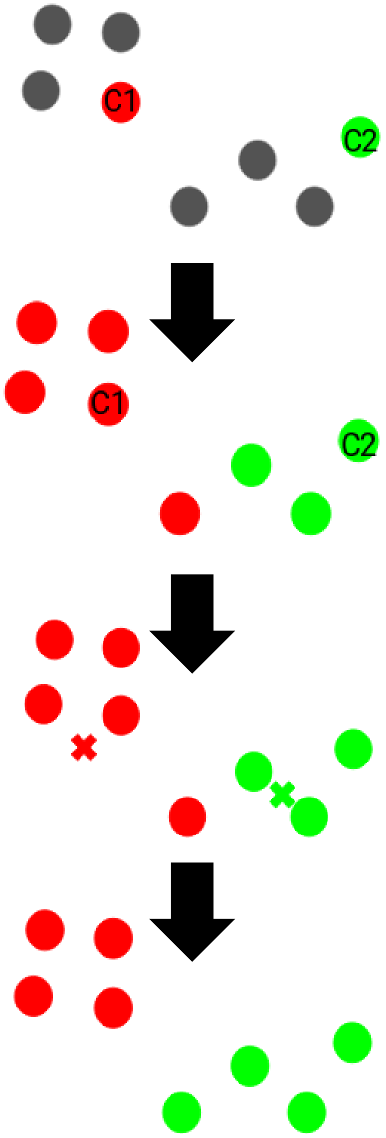


Samples in two-dimensional (2 features) space
AFTER clustering in three groups

The number of clusters (k) is a hyper parameter of clustering models: needs to be defined prior performing clustering

K-means: prominent clustering algorithm

- User provides the number of clusters K
- K-means iterative process involves the following steps:
  1. Selects K samples from data, or generates K points to be used as centroids (array of centroids is referred to as code book)
  2. Assigns all samples to the closest cluster centroid (referred to as mapping from code book)
  3. Recomputes the centroids of newly formed clusters
  4. Repeats steps 2 and 3



- Stopping criteria for K-means:
  - Centroids of newly formed clusters do not change
  - Samples remain in the same cluster
  - Maximum number of iterations is reached

K-means Issues

- Works only with numerical data
  - Encoding techniques needed to convert categorical data to numerical
- Distance-based algorithm: Euclidean distance
  - Features should be of similar scale – data scaling needed e.g. Standard scaler, Robust scaler, etc.
- Depends on initial centroid selection
  - The more optimal the positioning of these initial centroids, the fewer iterations of the *k*-means algorithm will be required for convergence
    - Strategic consideration to the initialization of these initial centroids could prove useful
  - Available initialization strategies:
    - Random selection: prone to bad selection (e.g. very close to each other)
    - K-means++ is a smart centroid initialization technique which selects centroids being as far as possible from one another

K-means bottom line

- Easy to implement and use
- Needs to scale features if in different scales
- Good initial centroid selection method available: K-means++
- Need to set K prior running the algorithm

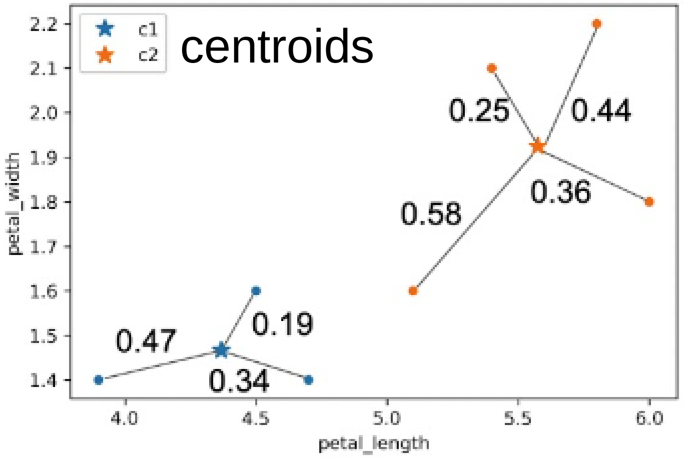
Choosing the best K (number of clusters)

- How can we determine the “best” value of *K?*
  - Is there an objective method?

- An estimation can be obtained using the following techniques:
  - Elbow method
  - Silhouette analysis

Elbow method parameters

- **Inertia**: The sum of squared distances* from each sample (data point) to its assigned cluster centroid
  - (*) Typically, the Euclidean distance metric is used
- **Distortion**: Weighted (by the cluster size) sum of squared distances from each sample (data point) to its assigned cluster centroid
- Example: use K-means, with K=2, to cluster 7 data points from a dataset having only 2 features in order to be able to visualize the distances in the 2-dimensional space and better understand the calculations below:

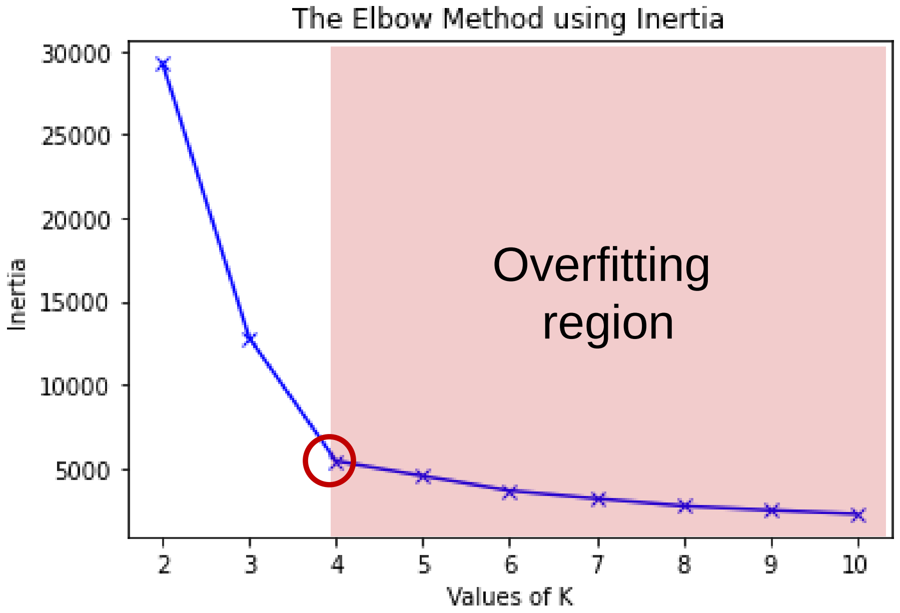


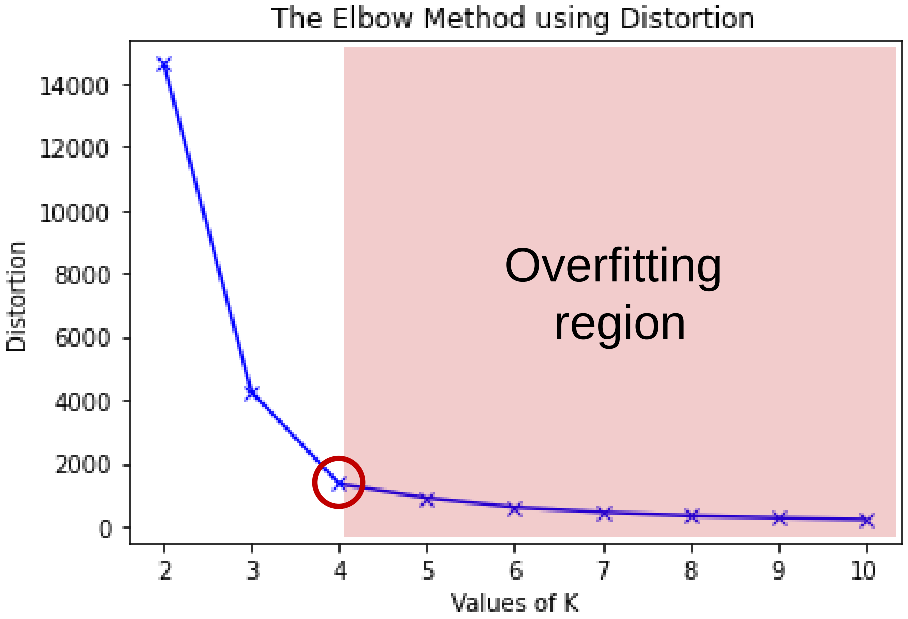
$$Inertia = 0.47^2 + 0.19^2 + 0.34^2 + 0.25^2 + 0.44^2 + 0.36^2 + 0.58^2$$

$$Distortion = \frac{(\,0.47^2 + 0.19^2 + 0.34^2)}{3} + \frac{(0.25^2 + 0.44^2 + 0.36^2 + 0.58^2)}{4}$$

Elbow method using inertia / distortion

- Run K-means algorithm for different values of K and plot the values of inertia and distortion for each iteration





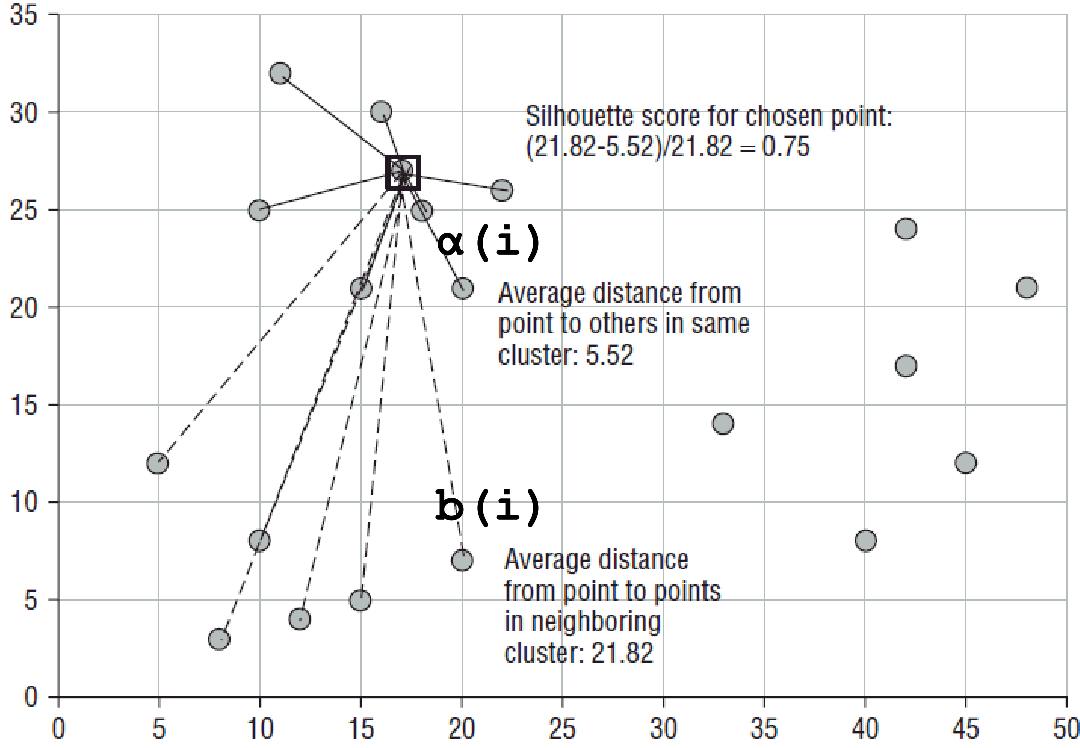
- “Best” number of clusters: value of K at the “elbow” i.e the point after which the distortion/inertia start decreasing in a linear fashion
  - adding another cluster doesn't give much better modeling of the data
  - smaller and tighter clusters explain less variation

Silhouette score

- Measures how similar a data point is to its own cluster (cohesion) compared to other clusters (separation)
  - Silhouette score for data point i :

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

  - ranges from -1 to 1
    - high value indicates that the data point is well matched to its own cluster and poorly matched to neighboring clusters
    - values near 0 indicate overlapping clusters
    - negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar
  - Find mean value of silhouette score of all data points => if most objects have a high value, then mean value is close to 1 and the clustering configuration is appropriate



Example: Drivers Dataset

- Includes 4000 drivers
- Each observation has 3 columns:
  - `Driver_ID`
  - `Distance_Feature`: mean distance covered per day
  - `Speeding_Feature`: mean percentage of time a driver was >5 mph over the speed limit

```
Driver_ID,Distance_Feature,Speeding_Feature
3423311935,71.24,28.0
3423313212,52.53,25.0
3423313724,64.54,27.0
```

- No target variable (no notion of groups, labels)
- Load dataset, drop Driver_ID column, scale features

```
dataset = pd.read_csv('fleet_data.csv')
dataset = dataset.drop(columns=['Driver_ID'])
scaler = RobustScaler()
X = scaler.fit_transform(dataset)
```

- Source code for all clustering experimentations can be found online

K-Means

- Python implementation: `sklearn.cluster.Kmeans()` class
- Run algorithm to define groups (clusters)

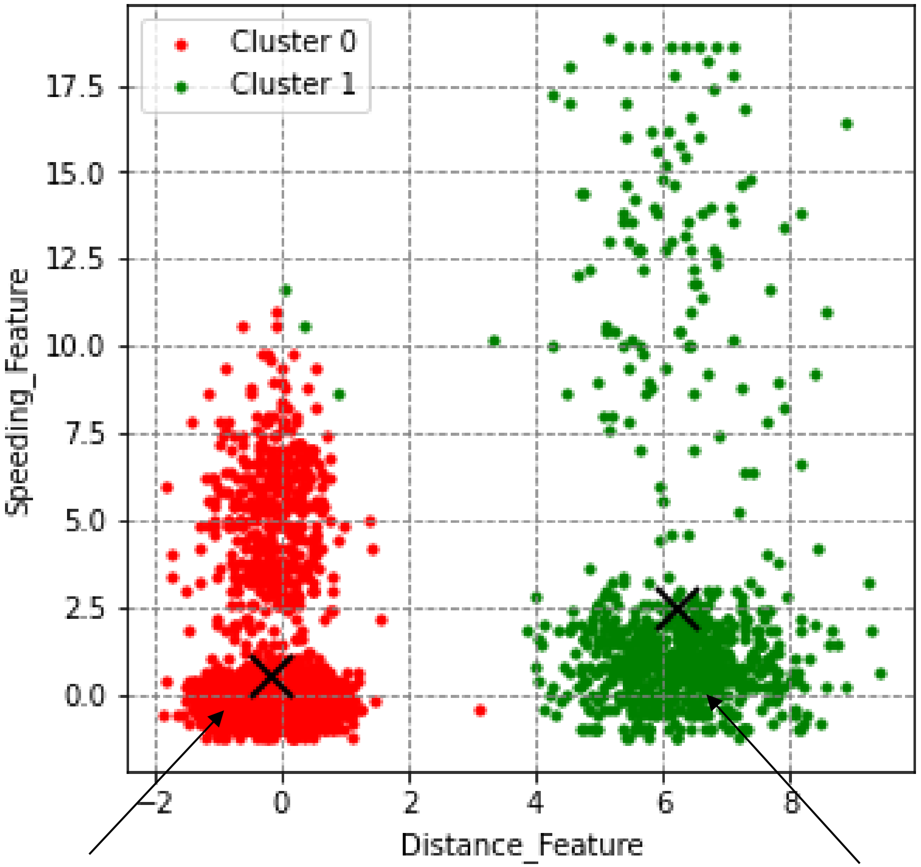
```
from sklearn.cluster import Kmeans
# create K-means object and run clustering on the input values (X)
# default k (n_clusters param) → 8
# default centroid initialization method (init param) → k-means++
kmeans = KMeans(n_clusters=2).fit(X)
print(kmeans.labels_) # labels of each sample
print(kmeans.centroids_)
```

- Assign new data samples to the most related cluster (closest centroid)

```
new_data = …
y_pred = kmeans.predict(new_data)
```

Visualizing results

- Run the *K*-means clustering algorithm for a range of *K* values
- Review the results

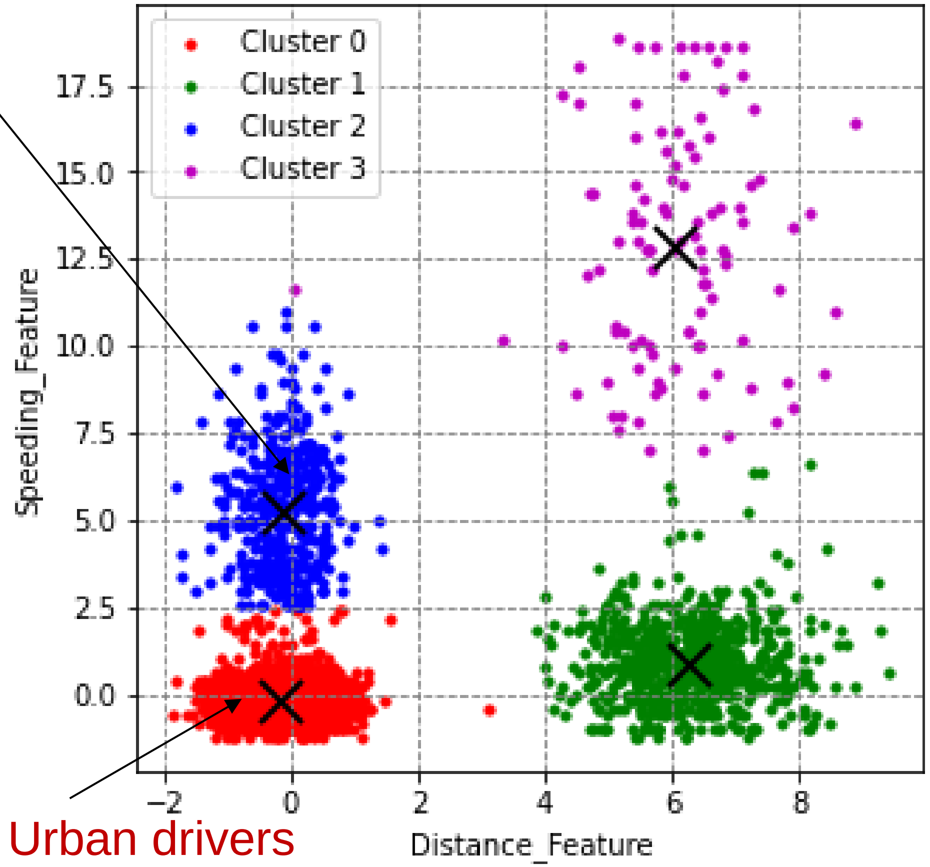


Urban drivers

K=2

Rural drivers

Urban drivers that are speeding frequently

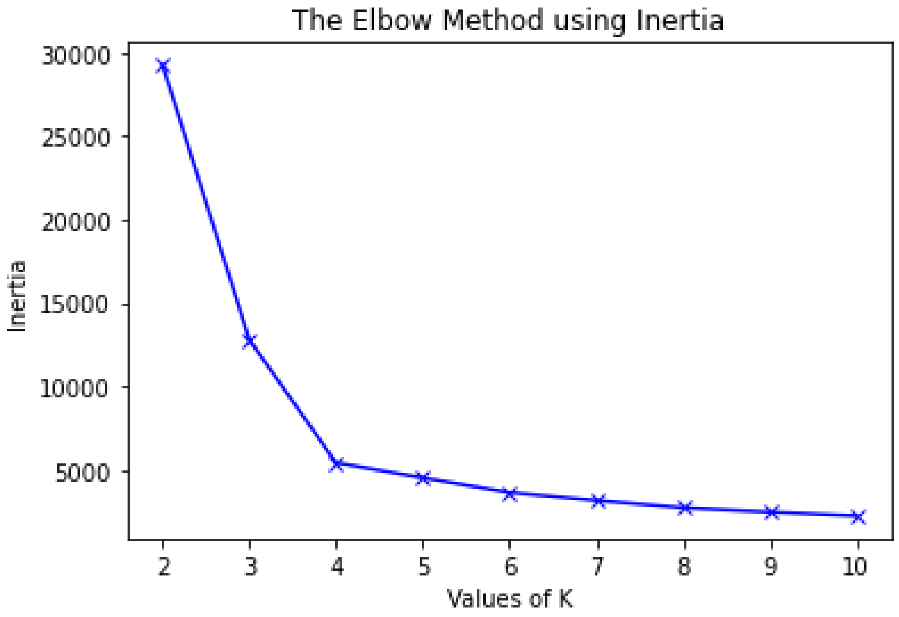


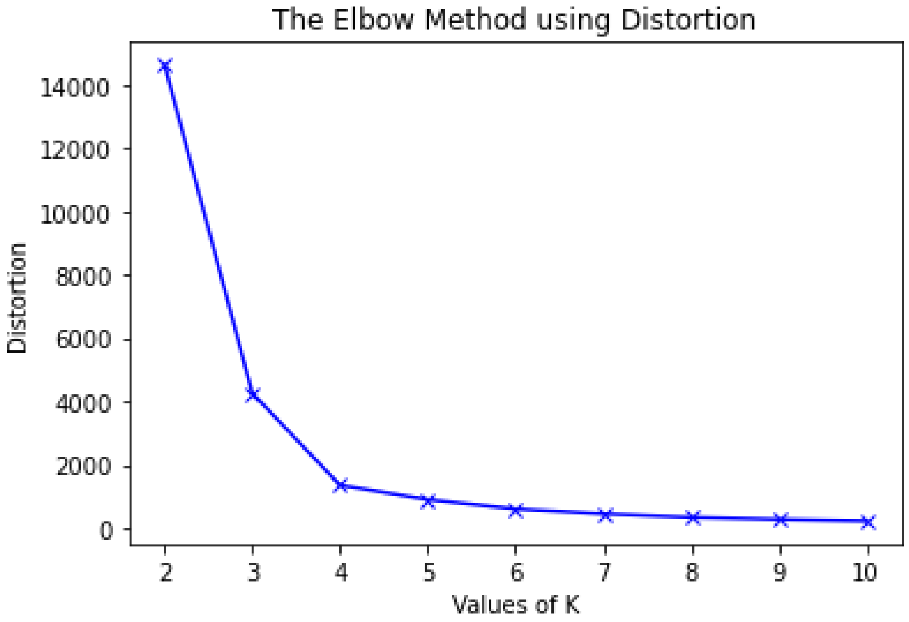
Urban drivers that follow speed limits

K=4

Elbow method

- We run the K-means algorithm for the values of k from 2 to 10 and plot the values of inertia and distortion for each iteration

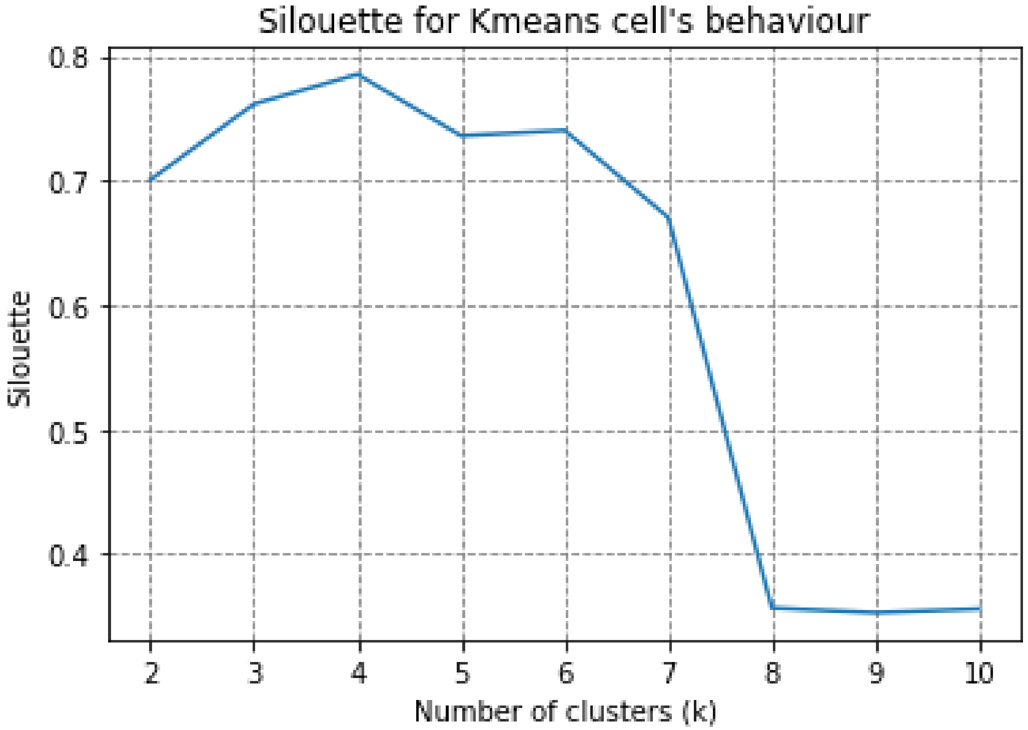




- It seems that the best number of clusters for grouping drivers is 4

Silhouette score

- We run the K-means algorithm for the values of k from 2 to 10 and plot the mean Silhouette score for each iteration



- Silhouette score confirms that the best number of clusters for grouping drivers is 4

Task: Wine Analysis

- Goal: Build a classifier to detect wine types
- Given dataset contains data of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars
- Analysis determined the quantities of 13 constituents (features) found in each of the three types of wines.
- Dataset snapshot:

```
class,alcohol,malic_acid,ash,alcalinity_of_ash,magnesium,total_...
1,14.23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065
1,13.2,1.78,2.14,11.2,100,2.65,2.76,.26,1.28,4.38,1.05,3.4,1050
1,13.16,2.36,2.67,18.6,101,2.8,3.24,.3,2.81,5.68,1.03,3.17,1185
```

Task: Wine Analysis

- Your analysis will answer the questions:
  - How many wines of each type are there in the dataset?
  - Which of the following classification algorithms:
    - Decision Trees, Random Forest, AdaBoost, Gradient Boosting, K-Nearest Neighbors, Gaussian Naïve Bayes, Support Vector Machines

  gives the best accuracy?
- Complete the given WineAnalysis.ipynb notebook file
  - Replace the keyword `None` with the appropriate source code based on the comments. There is also a question to answer. No worries if your results are slightly different than the results shown in the given notebook file.
- Submit the completed notebook file to Moodle by Tuesday 25th of March @ 09:00 am

Appendix – Problem with artificial balancing

- Let's say you're recognizing hand-written letters from English alphabet (26 letters). Overbalancing every letter appearance will give every letter a probability of being classified (correctly or not) roughly 1/26, so classifier will forget about actual distribution of letters in the original sample. And it's ok when classifier is able to generalize and recognize every letter with high accuracy.
- But if accuracy and most importantly generalization isn't "so high" (I can't give you a definition - you can think of it just as a "worst case") - the misclassified points will most-likely equally distribute among all letters, something like:
  - "A" was misclassified 10 times
  - "B" was misclassified 10 times
  - "C" was misclassified 11 times
  - "D" was misclassified 10 times
  - ...and so on

Appendix – Problem with artificial balancing

- As opposed to without balancing (assuming that "A" and "C" have much higher probabilities of appearance in text)
  - "A" was misclassified 3 times
  - "B" was misclassified 14 times
  - "C" was misclassified 3 times
  - "D" was misclassified 14 times
  - ...and so on
- So frequent cases will get fewer misclassifications. Whether it's good or not depends on your task. For natural text recognition, one could argue that letters with higher frequencies are more viable, as they would preserve semantics of the original text, bringing the recognition task closer to prediction (where semantics represent tendencies). But if you're trying to recognize something like screenshot of ECDSA-key (more entropy -> less prediction) - keeping data unbalanced wouldn't help. So, again, it depends.

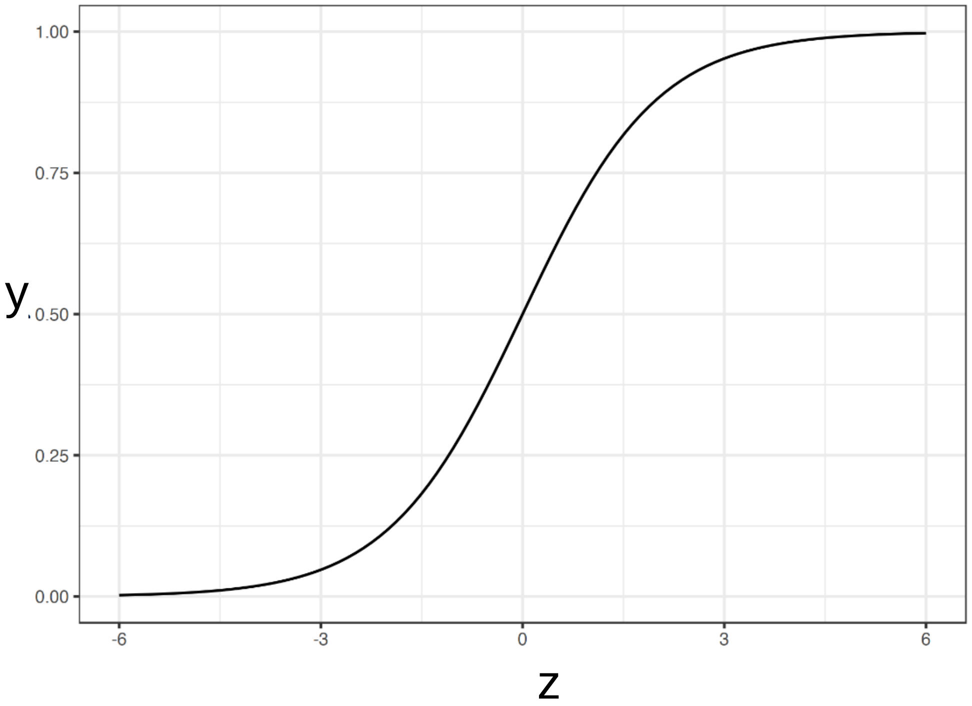
Appendix – Problem with artificial balancing

- **The most important distinction is that the accuracy estimate is, itself, getting biased** (as you can see in the balanced alphabet example), so you don't know how the model's behavior is getting affected by most rare or most frequent points.

Appendix – Logistic Regression

- Logistic regression name comes from the logistic sigmoid function

$$y = logistic(z) = \frac{1}{1 + e^{-z}}$$



- Logistic function outputs a value (a probability) between 0 and 1
  - Output y can be seen as the probability of belonging to the positive class, $P_{(y=1)}$
  - The returned probability can be converted to a binary category
    - $z \geq 0 \Rightarrow P_{(y=1)} \geq 0.5 \Rightarrow Sample\ belongs\ to\ positive\ class$ (e.g. spam)
    - $z < 0 \Rightarrow P_{(y=1)} < 0.5 \Rightarrow Sample\ belongs\ to\ negative\ class$ (e.g. not spam)
- Input z can be expressed as $z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots \beta_n X_n$, where Xi are independent variables (features) of the classification problem

Appendix – Logistic Regression Interpretation

- The interpretation of the coefficients (β_0, β_1, etc.) in logistic regression differs from the interpretation of the coefficients in linear regression
- Coefficients do not influence the probability linearly any longer
- Reformulate the equation so that only the linear term is on the right side of the formula

$$y = P_{(y=1)} = \frac{1}{1+e^{-z}} \Rightarrow \ln\left(\frac{P_{(y=1)}}{1-P_{(y=1)}}\right) = \ln\left(\frac{P_{(y=1)}}{P_{(y=0)}}\right) = z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots \beta_n X_n$$

- We call the term in the ln() function “odds” and wrapped in the logarithm it is called log odds
- This formula shows that the logistic regression model is a linear model for the log odds

Appendix – Logistic Regression Assumptions

$$y = \frac{1}{1 + e^{-z}} \Rightarrow \ln\left(\frac{y}{1 - y}\right) = z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots \beta_n X_n$$

- Linear relationships between X and ln(y)
- No or little multicollinearity
  - Multicollinearity: two or more of the independent variables are highly correlated to one another