MACHINE LEARNING PROJECT

Uranie JEAN-LOUIS

Appanna MACHIMANDA

MSc AgFood Data management

28th Feb 2021

Ponnappa Appanna MACHIMANDA
Uranie JEAN-LOUIS

Machine learning project

Abstract

Over the past few decades, the world has seen an increase in carbon dioxide emissions. “The average carbon footprint for a person in the United States is 16 tons, one of the highest rates in the world. Globally, the average is closer to 4 tons. To have the best chance of avoiding a 2℃ rise in global temperatures, the average global carbon footprint per year needs to drop under 2 tons by 2050” (The Nature Conservancy, 2021). Our project uses a data set found in Kaggle “Environmental Impact of Food Production”, that tracks the CO2 emissions for the production of various foods. With the availability of more data in the field of agriculture, we are now presented with the opportunity to optimise production and hence lower the emissions. In this project we use machine learning techniques to see if we can predict the type of food : carbohydrate, protein, or fat; based on CO2 emission generated from land use change, animal feed, farm, processing, transport, packaging and retail.

Introduction

The objective of this project is to try 4 machine learning methods on the dataset. The methods can be based on classification, clustering or regression. Since our dataset was unlabelled, clustering seems like a good option. However, we would not know how many clusters are appropriate and where to stop tuning the parameters. So we manually grouped the foods based on their type -- carbohydrate, protein or fat-- and decided to carry out classification techniques first. The methods used were K nearest neighbours, Random Forest, and Naïve Bayes. After these classification methods we used a clustering algorithm to verify if there was an intrinsic connection between the features and the target class.

Motivation

The dataset considered is composed of 7 variables corresponding to different stages of food production. Each variable describes the greenhouse gas emissions per kg of food product. The samples are of 43 unique foods. Using the machine learning methods we hope to identify if we can predict the class of our target column based on the features. To perform machine learning, 3 classification and 1 clustering method will be carried out.

Experiments

Preprocessing

- We manually classified the foods into 4 categories: Complex carbs, Simple carbs, Fats and Proteins.
- The original data set had many columns with a more granular level of detail. We have chosen only the high level CO2 categorizations based on the stage of farming: Land use change, animal feed, farm, processing, transport, packaging and retail. The remaining columns were dropped.
- Next a label encoder was used to convert the target categorical classes into numerical.

- Our target variable being categorical, a reencoding was made with the "labelEncoder" function from scikit learning pre-processing package. After target variable was coded as integer, the dataset was split into training and testing, with 70% for training

CV on Hyperparameters

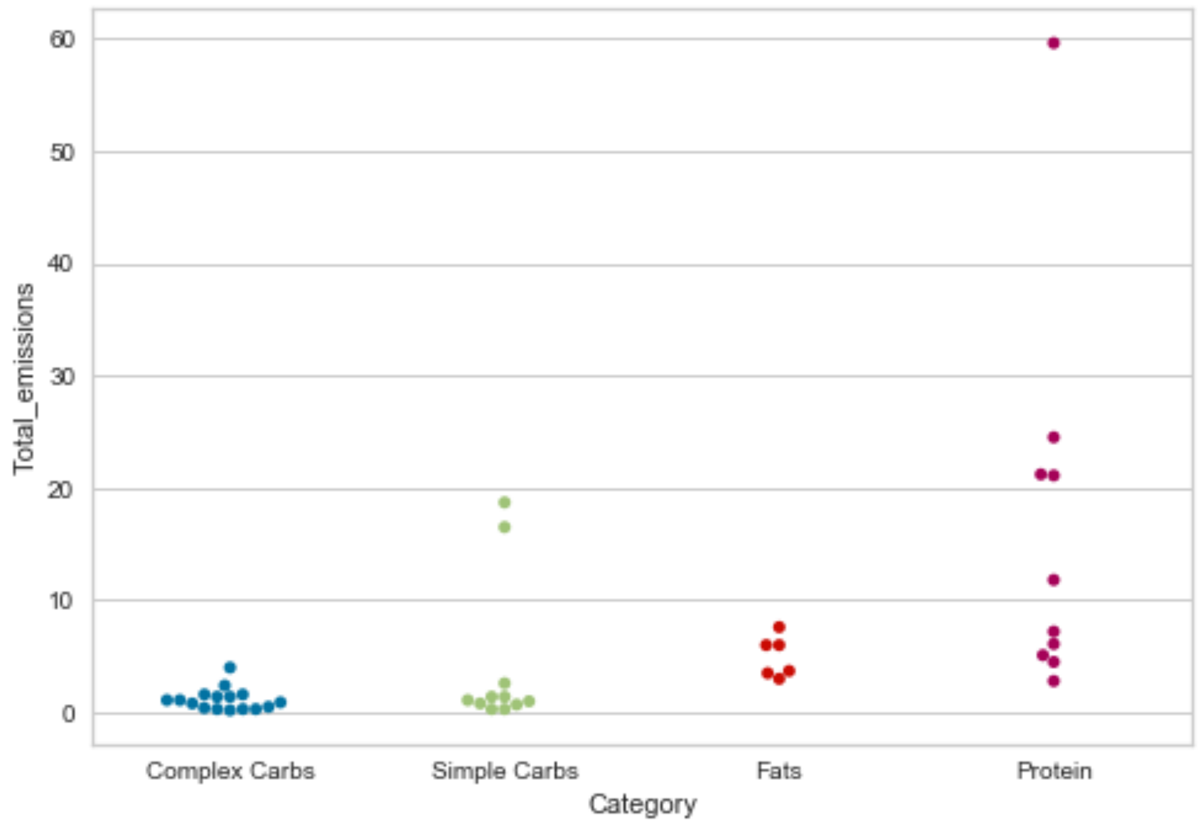
The typical method of judging the accuracy is by using the train test split method in which a certain percentage of the data is held out for testing purposes. However, in the train test split we might have variation in the results for different parts of the dataset. Cross Validation is done to ensure that the entire data set is represented while assessing the performance of the algorithm.

If cross validation is not done, we risk overfitting the algorithm to a particular part of the dataset. In this project we use grid search cross validation where we define a parameter grid and check the best hyperparameters to choose and the optimal results we can expect after cross validation.

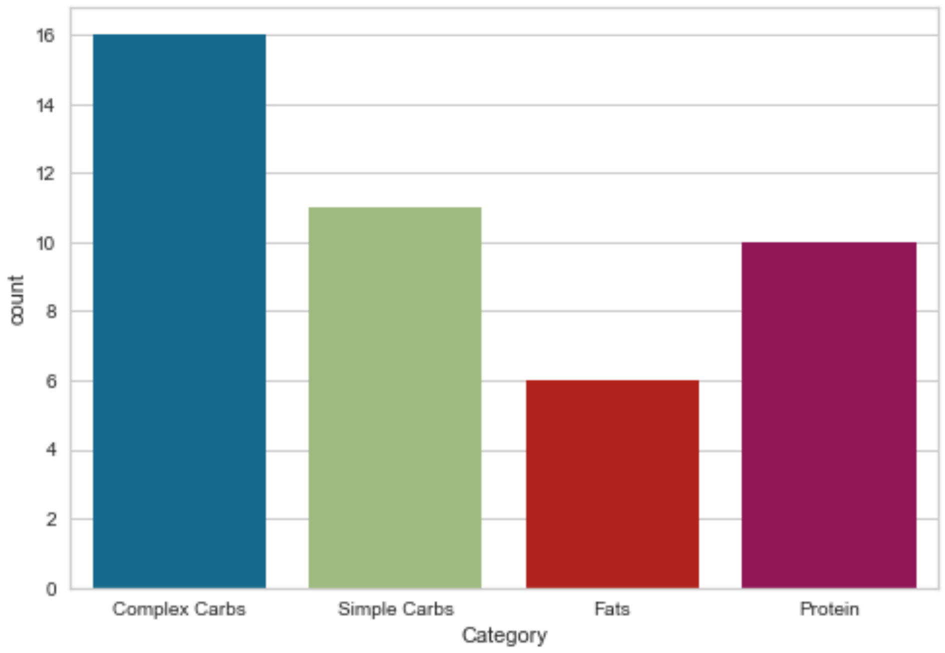
The stratified shuffle split method is used for selecting the training and test sets. This method selects the folds based on random samples, while keeping the proportion of the target class evenly distributed between the folds.

Data visualization

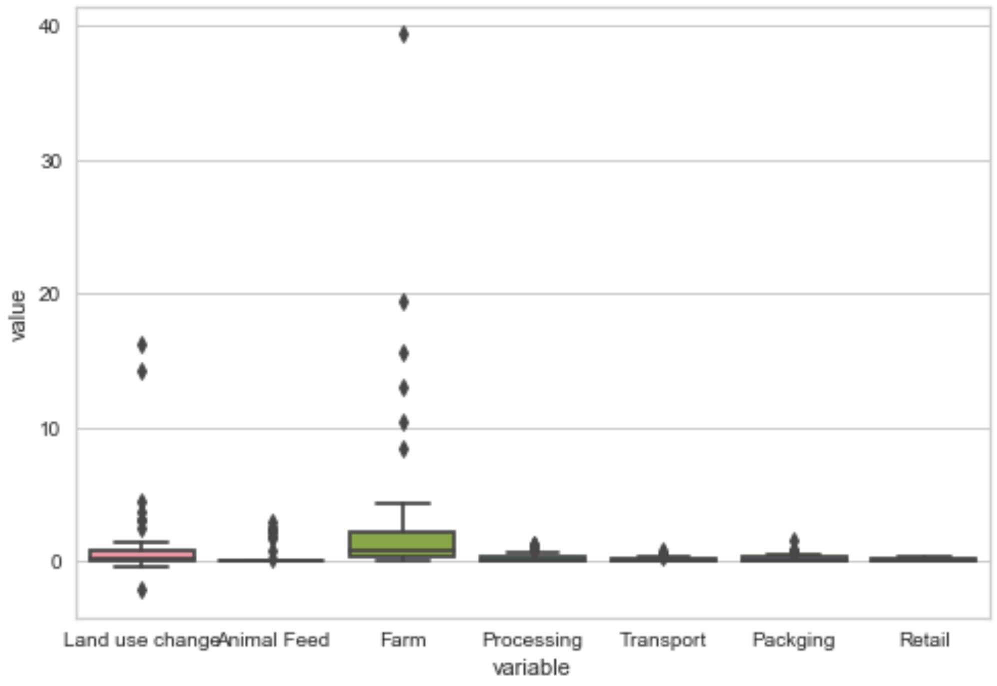
Distribution of the target classes in the dataset based on total emissions from the features considered:



We can see that complex carbs seem to have low emissions and are densely grouped. Next we have fats which are also relatively well grouped. For Simple carbs we can see 2 outliers and the rest are relatively well grouped. Proteins appear to have the widest distribution.



In the above graph we can see that most samples in the dataset are complex carbs. The Simple Carbs and Proteins account for almost the same number of samples. Whereas, Fats have the least representation in the dataset.



In terms of the distribution of the features, the above graph gives us an idea. We can see that 'Farm' emissions are the most widely distributed followed by 'Land use change'. The remaining features seem to have a denser distribution.

Classification

K Nearest neighbors

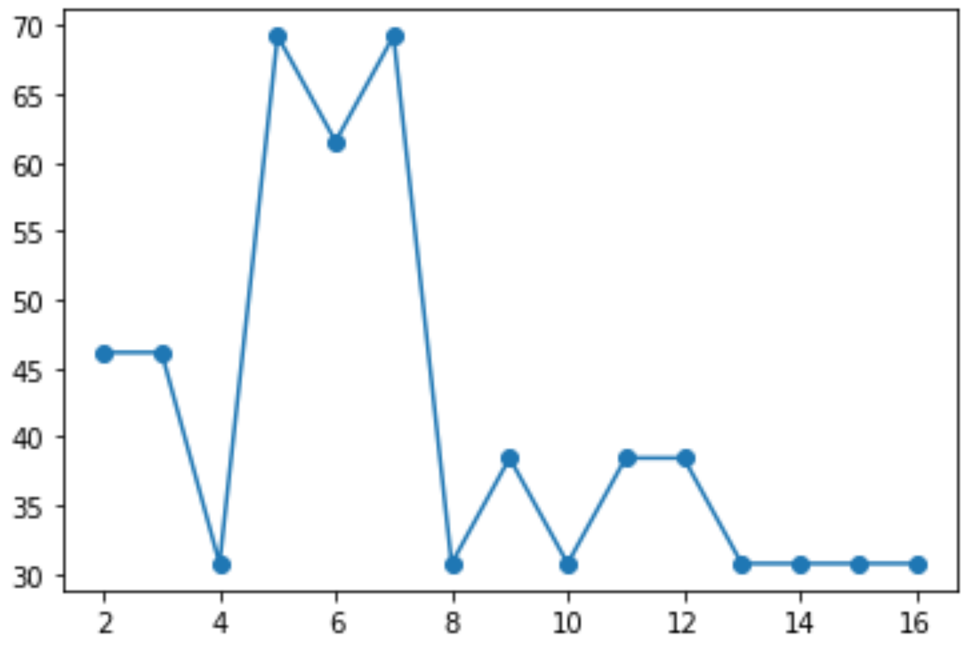
This is an intuitive method of machine learning where the distance to the neighbourhood zone is used to define the category of a new sample. The number of neighbors to consider is the main hyper parameter for this algorithm. The ‘weights’ parameter further dictates the impact of a neighbourhood on a test sample. In our case we use the default weight parameter ‘uniform’. This means that only the actual number of neighbors is considered and their distance to the test sample is not given additional weightage.

In our dataset, the physical units of measurement remain the same (CO2 emissions per kg of food product) and so we do not use scaling or normalizing.

Train-test-split

First we split the data into a training and test set in 70:30 ratio. This means we hold out 30% of the data exclusively for testing the accuracy of the model.

As we can see in the graph below, the lowest error is about 30% and occurs first at k=4 neighbors. Then the error rate increases and lowers again. After k=13 neighbors, the error remains constant at around 30%. This indicates an overfitting. An error of 30% implies an accuracy of 70%.



Cross validation

We will use cross validation techniques to ensure that the different sections of data do not adversely affect the scores for accuracy.

Typically the K-fold cross validation method is used where the data is divided into ‘k’ folds. For training the algorithm ‘k-1’ folds are used and testing is done on 1 fold. These folds apply in such a way that for every iteration a different test fold is selected.

In the grid search method, we will use the "stratified shuffle split" technique for cross validation. This is a variation of the K-fold method and preserves the percentage of classes in each fold. Further, the selection of the elements for each fold are randomly selected (not sequential) and this provides an added degree of error aggregation.

We have selected 5 iterations for re-shuffling and set the test size to be 30%.

To avoid ties in selection of the neigbourhood space, the parameter grid consists of odd values. The parameter grid is defined to test out different values for the hyper parameters.

Using this method, the best accuracy score for the data was 52.3% and the optimal number of neighbors to consider was 3.

We then use the optimal neighbour value of 3 and check how the algorithm performs. The classification report is described below:

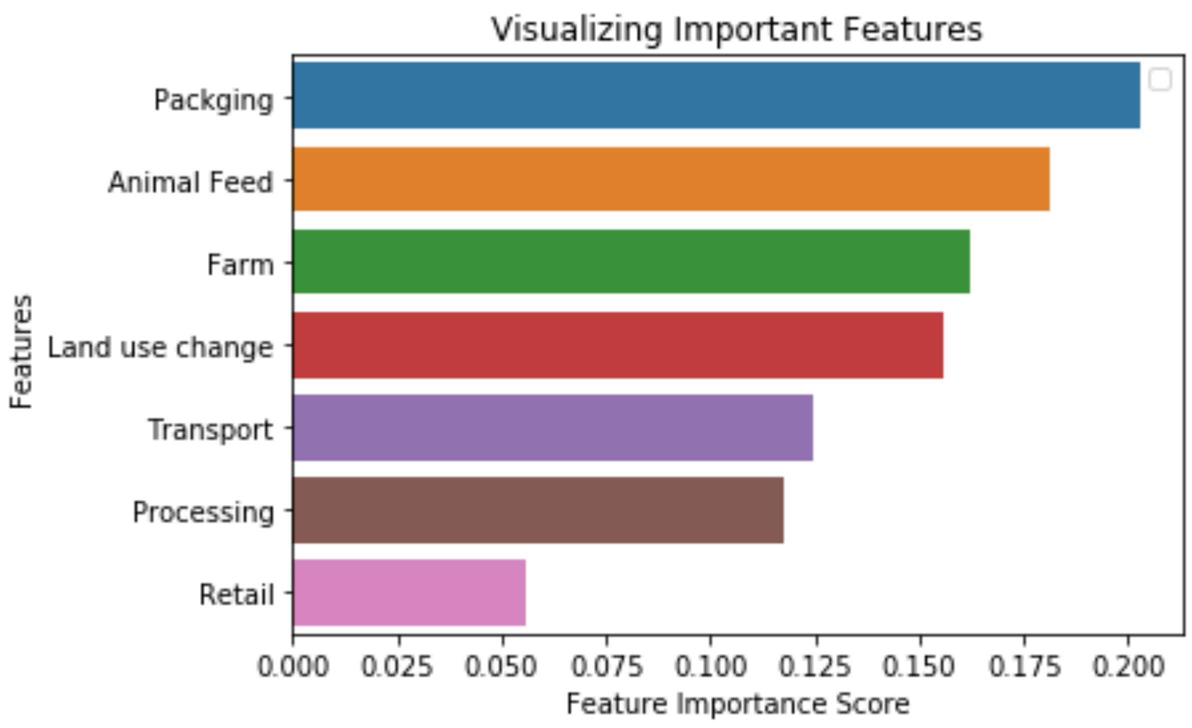
	precision	recall	f1-score	support
0	0.67	0.75	0.71	8
1	0.00	0.00	0.00	0
2	1.00	0.50	0.67	2
3	0.00	0.00	0.00	3
accuracy			0.54	13
macro avg	0.42	0.31	0.34	13
weighted avg	0.56	0.54	0.54	13

Random Forests

A random forest uses multiple decision trees for different sub samples of the dataset. The main hyper parameters we consider are number of trees (specified by n_estimator value), and the number of features.

In our case we use the whole dataset since the number of samples is less.
The parameter n_estimator indicates the number of decision trees in the forest. It was chosen by gridsearch cross validation method. The reshuffling parameters were the same as the knn methods and train-test-split method was also set at 30% test samples.
The optimal number of decision trees to consider was 100 obtaining an accuracy of 61.53%.

Plotting the significance of each of the features we get the following:



By selecting the top 3 features and preserving the number of estimators at 100, we get an accuracy of 76.9%

Naïve Bayes

Naïve Bayes algorithm is one of the quickest ways for classification. The algorithm assumes independence of each of the features in predicting the target class. It calculates the probability of a particular class based on the independent features. In a gaussian naïve bayes classifier, it is assumed that the features are in a Gaussian distribution.

Running the Gaussian Naïve Bayes classifier method on our data set, we get an accuracy score of 53.84%.

There are a total of 13 points where 4 are mislabelled.

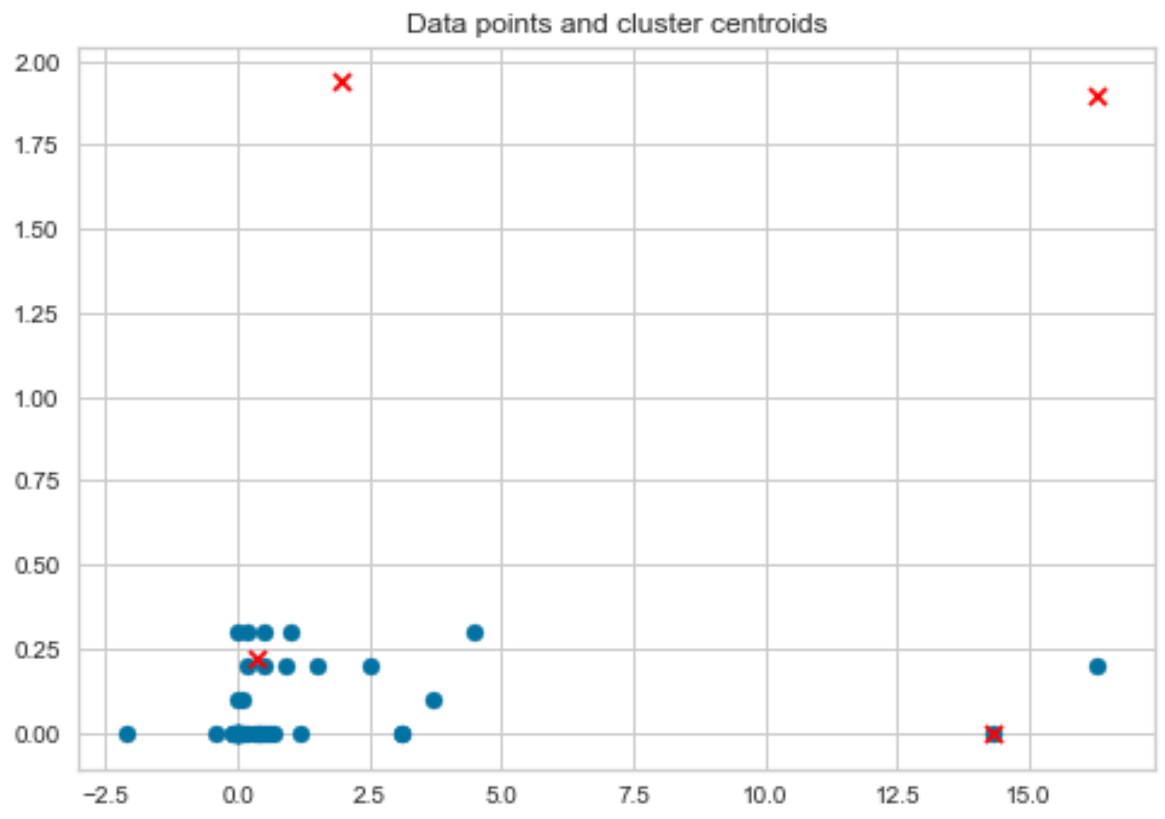
The classification achieved through this method is decent and is quick. However, the algorithm fails to account for connections between the features.

Clustering

K means clustering

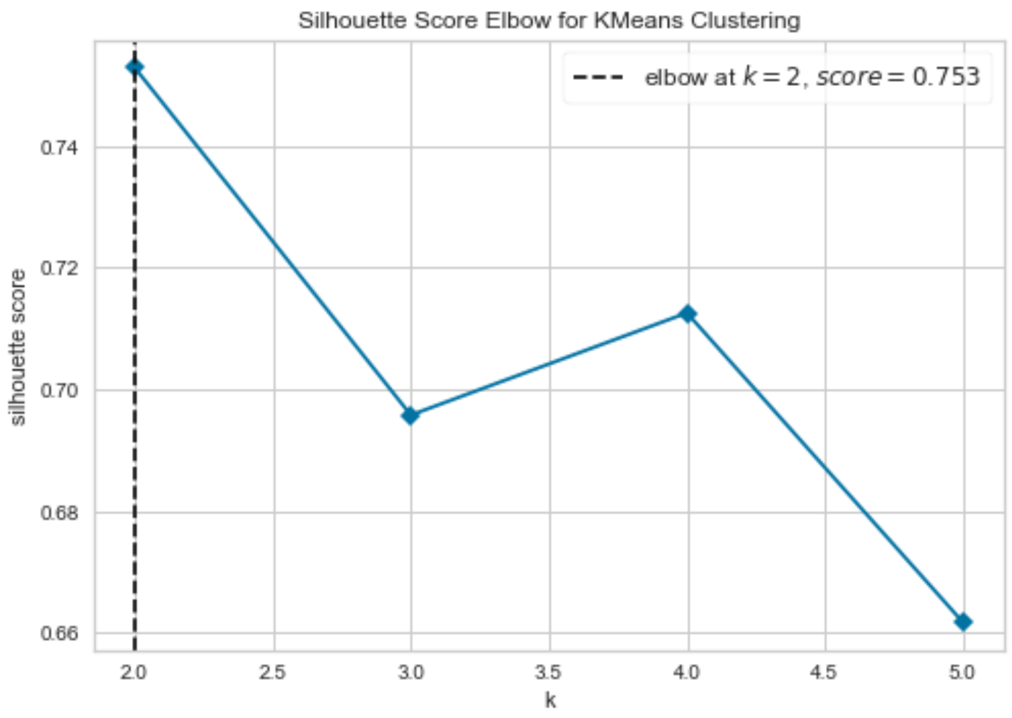
The k means clustering algorithm involves minimizing the within cluster distance and maximising the between cluster distance. This distance adjustment happens over several iterations. This is an unsupervised machine learning algorithm and the target labels are not known to the algorithm. Intuitively this algorithm is easier to understand and it makes sense that samples from similar classes can appear closer to each other in a multidimensional space.

In our dataset we set 4 clusters based on our manual classification of the samples into simple carbohydrates, complex carbohydrates, proteins and fats. The data that is fed into the algorithm is not informed of the target class.



In the above chart we see that 4 cluster centres are created but there are no samples close to 2 of these centres. Most of the samples surround only 1 cluster. This indicates that the features are not very correlated with the target classification.

To evaluate the performance of the model, two methods can be considered. The first one can be used when the label information are not known and this method is based on the silhouette score. It can be plotted to check for which values of k, the separation of the clusters are maximum. Using the elbow method we see that for k=2, the score is 75.3%. The closer this percentage is to 100%, the more is the separation of the clusters. If the percentage is closer to 0, then the clusters are likely to be bordering the neighbouring clusters.



The second method to consider is using the 4 labels that we defined and comparing the similarity with the labels made with the clustering. In this case, using the adjust rand index method the accuracy of the clustering algorithm with our dataset is 6.6%. However, if we used other technics such as normalized mutual information the accuracy is at 2%, with adjusted mutual information, 0.8% and with Fowlkes-Mallows, 4%. Even if the scores are different all the results are below 50% of accuracy which indicate a lack of intrinsic correlation between the features and the target class.

Conclusion

Although the accuracy in classification methods were above 50%, it is still not sufficient to make informed decisions about the type of food based on the CO2 emissions. This low accuracy could be attributed to the few samples available or a disconnect between the target and features. Among the 3 classification methods, random forest was the most accurate one mainly due to its versatile modelling capabilities. However, the poor performance of the clustering algorithm further confirmed our suspicion that the classes were not intrinsically connected to the features.

We can conclude that the classification models can be fit on any data even though they are not intuitively related. Hence, we must double check our methods (in our case classification was cross verified with clustering) and be wary about jumping to conclusions regarding connections between the target and features. Overfitting the data for results which look nice is not a good idea.

Future work

For further analysis, the number of samples could be increased to study the extent to which accuracy can improve. Further research could be directed to explore if our features can link better with more intuitively relevant target classes.

Reference

https://www.nature.org/en-us/get-involved/how-to-help/carbon-footprint-calculator/
[Accessed Feb, 28 2021]
https://ourworldindata.org [Accessed Feb, 28 2021]
https://scikit-learn.org/stable/index.html [Accessed Feb, 28 2021]

Python Code

```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA
import numpy as np
import matplotlib .pyplot as plt
from matplotlib .colors import ListedColormap
from sklearn import neighbors , datasets
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import (train_test_split, GridSearchCV,
                                     StratifiedShuffleSplit)
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix

###Impot dataset

food_CO2 = pd.read_csv('Food_Production_2.csv', delimiter=';')


###Removing column that we don't need

food_CO2.drop(food_CO2.iloc[:,11:], axis = 1, inplace=True)


####Keeping all features
X = food_CO2[['Land use change', 'Animal Feed', 'Farm', 'Processing', 'Transport',
           'Packging', 'Retail']].values

###Reencoding

le = LabelEncoder()

food_CO2['Category_num']= le.fit_transform(food_CO2['Category'])

y = food_CO2["Category_num"].astype(int).values
```

```
####Splitting data

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=5)




#####CLASSIFICATION#######



########K-nearest neigbour###########

#Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors=2)

#Train the model using the training sets
knn.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = knn.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

####Error

error = 1 - knn.score(X_test, y_test)
print('Erreur: %f' % error)

####Find the best k


####Cross Validatation method

# Only odd numbers, to prevent ties
param_grid = {'n_neighbors': np.arange(1, 20, 2)}


knn = KNeighborsClassifier()

# Perform grid search with cross-validation
ss = StratifiedShuffleSplit(n_splits=5, test_size=.3, random_state=0)
gscv = GridSearchCV(knn, param_grid, cv=ss)
gscv.fit(X, y)
```

```
print("Best params:", gscv.best_params_)
print("Best score:", gscv.best_score_)



# Print out confusion matrix
cmat = confusion_matrix(y_test, y_pred)
#print(cmat)
print('TP - True Negative {}'.format(cmat[0,0]))
print('FP - False Positive {}'.format(cmat[0,1]))
print('FN - False Negative {}'.format(cmat[1,0]))
print('TP - True Positive {}'.format(cmat[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cmat[0,0],cmat[1,1]]),np.sum(cmat))))
print('Misclassification Rate:
{}'.format(np.divide(np.sum([cmat[0,1],cmat[1,0]]),np.sum(cmat))))


# Retrain model using optimal k-value
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)

# Print out classification report and confusion matrix
print(classification_report(y_test, pred))



#######Random Forest#########

from sklearn.ensemble import RandomForestClassifier

# Feature Scaling
#from sklearn.preprocessing import StandardScaler

#sc = StandardScaler()
#X_train = sc.fit_transform(X_train)
#X_test = sc.transform(X_test)

rfc=RandomForestClassifier(random_state=0)

param_grid = {
   'n_estimators': [100, 150, 200, 250, 300, 350, 400, 450, 500]
}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= ss)
CV_rfc.fit(X_train, y_train)


CV_rfc.best_params_
```

```
print("Best score:", CV_rfc.best_score_)

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred_clf=clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_clf))


####Selecting features

feature = food_CO2.iloc[:,3:10]

feature_imp =
pd.Series(clf.feature_importances_,index=feature.columns).sort_values(ascending=False)
feature_imp

#Plot

import seaborn as sns

# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()


###Generating model on selected features

X_new = food_CO2[['Animal Feed','Farm','Packging']]

X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(X_new, y,
                                                    test_size=0.30,
                                                    random_state=5)



#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
```

```
clf.fit(X_train_new,y_train_new)

y_pred_clf=clf.predict(X_test_new)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test_new, y_pred_clf))



#####Naive Bayes######


#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
gnb = GaussianNB()

# Train the model using the training sets
gnb.fit(X_train, y_train)

#Predict Output
y_pred_gnb= gnb.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_gnb))


######## CLUSTERING #########

###K means

from sklearn.cluster import KMeans


X_clust = food_CO2.copy()

##Keeping all the features
X_clust.drop(columns=['Category', 'Sub_category', 'Food product'], inplace=True)


#Cluster K-means

model=KMeans(n_clusters=4, random_state = 0)

#adapter le modèle de données
model.fit(X_clust)

# Plotting the cluster centers and the data points on a 2D plane
plt.scatter(X[:, 0], X[:, -1])

plt.scatter(model.cluster_centers_[:, 0], model.cluster_centers_[:, 1], c='red', marker='x')
plt.title('Data points and cluster centroids')
plt.show()


# Calculate silhouette_score
from sklearn.metrics import silhouette_score

print(silhouette_score(X, model.labels_))


# Import the KElbowVisualizer method
from yellowbrick.cluster import KElbowVisualizer

# Instantiate a scikit-learn K-Means model
model = KMeans(random_state=0)

# Instantiate the KElbowVisualizer with the number of clusters and the metric
visualizer = KElbowVisualizer(model, k=(2,6), metric='silhouette', timings=False)

# Fit the data and visualize
visualizer.fit(X_clust)
visualizer.poof()


#####Performance of our clustering

labels = le.fit_transform(food_CO2['Category'])
print(labels)
print(model.labels_)


##Adjusted Rand Index
metrics.adjusted_rand_score(model.labels_, labels)
#0.06

#Fowlkes-Mallows Score
from sklearn.metrics.cluster import fowlkes_mallows_score
fowlkes_mallows_score (labels, model.labels_)
#0.4

#Mutual Information Based Score

#Normalized Mutual Information (NMI)
from sklearn.metrics.cluster import normalized_mutual_info_score
normalized_mutual_info_score (labels, model.labels_)
#0.2

##Adjusted Mutual Information (AMI)
from sklearn.metrics.cluster import adjusted_mutual_info_score
```

```
adjusted_mutual_info_score (labels, model.labels_)
#0.08

### Data visualization
import seaborn as sns
sns.set_style('whitegrid')

#swarm plot
sns.swarmplot(data=food_CO2,x='Category', y='Total_emissions')

#Count plot
sns.countplot(x='Category',data=food_CO2)

#Box plot of features
box = pd.melt(food_CO2,id_vars=['Category'], value_vars=['Land use change', 'Animal Feed',
'Farm', 'Processing', 'Transport',
                        'Packging', 'Retail'])
sns.boxplot(data=box,x='variable', y='value')
```