ASSIGNMENT 3
DATA MINING

CHRISTOS GOULAS 2677 - IOANNA PAPAGIANNI 2790

QUESTION 1

For this analysis we use Jupyter Notebook and the file is in IPython format. We downloaded "train.json"
a dataset with recipes from Kaggle and both this file and "classifiers.ipynb"
exist in the same folder.

Analysis 1: Compare Classifier metrics CountVectorizer VS TfidfVectorizer

We work with the category " italian " and "mexican" at first, in order to make classifiers that differentiates different cuisines.
We extract features with two different techniques, **CountVectorizer** and **TfidfVectorizer** and compare the results for metrics such as **mean accuracy score, mean precision score, mean recall score, mean f1-measure score** and **mean confusion-matrix**.
The classifiers we use for this experiment are: **Logistic Regression, SVM, Decision Trees, K-NN, Naïve Bayes** and we use 5-Fold Cross Validation to evaluate our prediction results.

```
****Logistic Regression mean scores COUNTVECTORIZER****

scores: {'fit_time': array([2.5504601 , 2.18405247, 2.21158576, 1.94333792, 1.90349221]),
'score_time': array([0.02898192, 0.02722764, 0.02730107, 0.02709079, 0.02767682]),
'test_accuracy': array([0.96288515, 0.96672504, 0.97162872, 0.96882662, 0.96812609]),
'test_precision_weighted': array([0.96309073, 0.9668767 , 0.97170751, 0.96893784, 0.96833373]),
'test_recall_weighted': array([0.96288515, 0.96672504, 0.97162872, 0.96882662, 0.96812609]),
'test_f1_weighted': array([0.96283275, 0.96668534, 0.97160427, 0.96879444, 0.96808231])}

mean score test accuracy: 0.967638326784304

mean score test precision weighted: 0.9677893006690403

mean score test recall weighted: 0.967638326784304

mean score test f1-measure weighted: 0.9675998230499061

Confusion Matrix: [[7683  155]
                   [ 307 6131]]

****Logistic Regression mean scores TFIDFVECTORIZER****

scores: {'fit_time': array([0.52974987, 0.52992105, 0.51307178, 0.4936018 , 0.5006001 ]),
'score_time': array([0.01318645, 0.00974274, 0.00974846, 0.00980425, 0.00980759]),
'test_accuracy': array([0.99264706, 0.99369527, 0.99194396, 0.99579685, 0.99334501]),
'test_precision_weighted': array([0.99274423, 0.99376687, 0.99206049, 0.99582877, 0.99342468]),
'test_recall_weighted': array([0.99264706, 0.99369527, 0.99194396, 0.99579685, 0.99334501]),
'test_f1_weighted': array([0.9926413 , 0.99369109, 0.99193702, 0.99579502, 0.9933403 ])}

mean score test accuracy: 0.993485628927578

mean score test precision weighted: 0.9935650103027747

mean score test recall weighted: 0.993485628927578

mean score test f1-measure weighted: 0.9934809442666724

Confusion Matrix: [[7838    0]
                   [  93 6345]]
```

ASSIGNMENT 3
DATA MINING

****SVM mean scores COUNTVECTORIZER****

```
scores: {'fit_time': array([0.4891715 , 0.49625278, 0.49789715, 0.5144918 , 0.49009752]),
'score_time': array([0.02744508, 0.02796507, 0.02731681, 0.02739477, 0.02725601]),
'test_accuracy': array([0.96113445, 0.96322242, 0.9614711 , 0.96532399, 0.96497373]),
'test_precision_weighted': array([0.96116578, 0.96322223, 0.9614711 , 0.96531882, 0.96497238]),
'test_recall_weighted': array([0.96113445, 0.96322242, 0.9614711 , 0.96532399, 0.96497373]),
'test_f1_weighted': array([0.96110708, 0.96320787, 0.9614711 , 0.96532033, 0.96496109])}

mean score test accuracy: 0.9632251394428174

mean score test precision weighted: 0.9632300630213381

mean score test recall weighted: 0.9632251394428174

mean score test f1-measure weighted: 0.9632134975948101

Confusion Matrix: [[7597  241]
                   [ 284 6154]]
```

****SVM mean scores TFIDFVECTORIZER****

```
scores: {'fit_time': array([0.19516706, 0.19173741, 0.19179201, 0.19114351, 0.19552565]),
'score_time': array([0.0107379 , 0.00990653, 0.00981736, 0.00994205, 0.00989962]),
'test_accuracy': array([0.99614846, 0.99649737, 0.9943958 , 0.9971979 , 0.99579685]),
'test_precision_weighted': array([0.99617529, 0.99651958, 0.99445244, 0.9972056 , 0.99582877]),
'test_recall_weighted': array([0.99614846, 0.99649737, 0.9943958 , 0.9971979 , 0.99579685]),
'test_f1_weighted': array([0.99614693, 0.99649612, 0.99439252, 0.9971973 , 0.99579502])}

mean score test accuracy: 0.9960072750641412

mean score test precision weighted: 0.9960363379540353

mean score test recall weighted: 0.9960072750641412

mean score test f1-measure weighted: 0.9960055765255399

Confusion Matrix: [[7837    1]
                   [  56 6382]]
```

****Decision-Tree mean scores COUNTVECTORIZER****

```
scores: {'fit_time': array([11.39658332, 33.20916319, 10.85297585, 11.07879996, 16.97399926]),
'score_time': array([0.02721238, 0.02717781, 0.02638102, 0.02665997, 0.02704692]),
'test_accuracy': array([0.93487395, 0.94535902, 0.93134851, 0.93870403, 0.94781086]),
'test_precision_weighted': array([0.93497916, 0.94541877, 0.9313223 , 0.93869318, 0.94793324]),
'test_recall_weighted': array([0.93487395, 0.94535902, 0.93134851, 0.93870403, 0.94781086]),
'test_f1_weighted': array([0.93490394, 0.94537721, 0.93131879, 0.93869756, 0.9477437 ])}

mean score test accuracy: 0.9396192732784883

mean score test precision weighted: 0.9396693301209688

mean score test recall weighted: 0.9396192732784883

mean score test f1-measure weighted: 0.9396082410941424

Confusion Matrix: [[7420  418]
                   [ 436 6002]]
```

****Decision-Tree mean scores TFIDFVECTORIZER****

```
scores: {'fit_time': array([2.93528199, 2.91533303, 2.83061743, 2.92146993, 2.83942223]),
'score_time': array([0.01470017, 0.01477718, 0.01456809, 0.01467276, 0.01457882]),
'test_accuracy': array([0.99894958, 0.99824869, 0.99754816, 0.99894921, 0.99859895]),
'test_precision_weighted': array([0.99895159, 0.99825055, 0.99755907, 0.99895122, 0.99860251]),
'test_recall_weighted': array([0.99894958, 0.99824869, 0.99754816, 0.99894921, 0.99859895]),
'test_f1_weighted': array([0.99894947, 0.9982485 , 0.99754755, 0.9989491 , 0.99859875])}

mean score test accuracy: 0.9984589177177

mean score test precision weighted: 0.9984629859026336

mean score test recall weighted: 0.9984589177177

mean score test f1-measure weighted: 0.9984586754890283

Confusion Matrix: [[7837    1]
                   [  19 6419]]
```

****k-NN mean scores COUNTVECTORIZER****

ASSIGNMENT 3
DATA MINING

```
scores: {'fit_time': array([4.22745609, 4.12844706, 4.11732197, 4.16310835, 4.35632157]),
'score_time': array([173.40004349, 168.35602403, 170.35519505, 172.46086407, 171.29579973]),
'test_accuracy': array([0.88830532, 0.88406305, 0.88231173, 0.88721541, 0.87845884]),
'test_precision_weighted': array([0.88823164, 0.88608528, 0.88566209, 0.89111778, 0.884777  ]),
'test_recall_weighted': array([0.88830532, 0.88406305, 0.88231173, 0.88721541, 0.87845884]),
'test_f1_weighted': array([0.8881997 , 0.88333008, 0.88132628, 0.8861979 , 0.87695044])}

mean score test accuracy: 0.884070871781287

mean score test precision weighted: 0.8871747597288895

mean score test recall weighted: 0.884070871781287

mean score test f1-measure weighted: 0.8832008807972722

Confusion Matrix: [[7345  493]
                  [1162 5276]]
```

****k-NN mean scores TFIDFVECTORIZER****

```
scores: {'fit_time': array([2.26060057, 2.25794148, 2.25653839, 2.29463649, 2.26201057]),
'score_time': array([85.24381018, 85.42045975, 85.69957519, 85.78246546, 85.19833612]),
'test_accuracy': array([0.93277311, 0.92924694, 0.9352014 , 0.94500876, 0.93309982]),
'test_precision_weighted': array([0.94010695, 0.93732614, 0.94204372, 0.95001376, 0.94036414]),
'test_recall_weighted': array([0.93277311, 0.92924694, 0.9352014 , 0.94500876, 0.93309982]),
'test_f1_weighted': array([0.93200522, 0.9283789 , 0.93450048, 0.94453364, 0.9323391 ])}

mean score test accuracy: 0.9350660053863928

mean score test precision weighted: 0.9419709424977883

mean score test recall weighted: 0.9350660053863928

mean score test f1-measure weighted: 0.9343514707413195

Confusion Matrix: [[7838    0]
                  [ 927 5511]]
```

****Naive-Bayes mean scores COUNTVECTORIZER****

```
scores: {'fit_time': array([0.46336746, 0.48547864, 0.46542311, 0.45993423, 0.46202064]),
'score_time': array([0.04326558, 0.02981758, 0.0301919 , 0.02966714, 0.02948952]),
'test_accuracy': array([0.96358543, 0.96462347, 0.96497373, 0.96672504, 0.96532399]),
'test_precision_weighted': array([0.96392113, 0.96502008, 0.96547506, 0.96698809, 0.96576285]),
'test_recall_weighted': array([0.96358543, 0.96462347, 0.96497373, 0.96672504, 0.96532399]),
'test_f1_weighted': array([0.96352096, 0.96455609, 0.96489876, 0.96667345, 0.96525427])}

mean score test accuracy: 0.9650463337699353

mean score test precision weighted: 0.9654334437437194

mean score test recall weighted: 0.9650463337699353

mean score test f1-measure weighted: 0.9649807059388958

Confusion Matrix: [[7704  134]
                  [ 365 6073]]
```

****Naive-Bayes mean scores TFIDFVECTORIZER****

```
scores: {'fit_time': array([0.10076237, 0.09514427, 0.09485245, 0.09402418, 0.09432292]),
'score_time': array([0.01139355, 0.01068044, 0.01070833, 0.01060581, 0.01093888]),
'test_accuracy': array([0.9772409 , 0.97478109, 0.97688266, 0.9765324 , 0.97618214]),
'test_precision_weighted': array([0.9781468 , 0.97588893, 0.97781698, 0.97749407, 0.97717212]),
'test_recall_weighted': array([0.9772409 , 0.97478109, 0.97688266, 0.9765324 , 0.97618214]),
'test_f1_weighted': array([0.9771774 , 0.9747017 , 0.97681713, 0.97646426, 0.97611174])}

mean score test accuracy: 0.9763238360142656

mean score test precision weighted: 0.9773037810783135

mean score test recall weighted: 0.9763238360142656

mean score test f1-measure weighted: 0.9762544433551351

Confusion Matrix: [[7838    0]
                  [ 338 6100]]
```

CountVectorizer provides a simple way to keep every ingredient as a whole word and keeps 1 if the specific word exists in the recipe else 0, whereas TfidVectorizer calculates term frequency

(TF) of the ingredients in each recipe without keeping the text of the ingredient as a whole word. For example, CountVectorizer would keep ‘jalapenosauce’ but TfidfVectorizer would keep ‘jalapeno’, ‘sauce’. CountVectorizer in our case is in binary form, TfidfVectorizer has in each vector the numbers (weights) represent features tf-idf score.

As we can see the TfidVectorizer performs better in all classifiers.

Analysis 2: Compare Classifier metrics VS Clustering metrics (both with CountVectorizer

In the second experiment, we work with the category “moroccan”, “filipino” and “british”, implementing the above classifiers and comparing the results with those of **agglomerative clustering** and **k-means** that we computed in a previous project on the same dataset (https://github.com/ioannapap/clustering).
The prior difference between classification and clustering is that classification is used in supervised learning technique where predefined labels are assigned to instances by properties and we want to know which class a new object belongs to, on the contrary, clustering is used in unsupervised learning where similar instances are grouped, based on their features or properties.

```
clustering = AgglomerativeClustering(n_clusters = 3, linkage='ward').fit(df)
print(np.unique(clustering.labels_))
```

```
[0 1 2]
```

```
def cluster_class_mapping(predicted,true_labels):
    C = metrics.confusion_matrix(predicted,true_labels)
    mapping = list(np.argmax(C,axis=1)) #for each row (cluster) find the best␣,→class in the co
nfusion matrix
    mapped_kmeans_labels = [mapping[l] for l in predicted]
    C2 = metrics.confusion_matrix(clustering.labels_,actual)
    return mapped_kmeans_labels,C2


mapped_agglo_labels,C = cluster_class_mapping(clustering.labels_,actual)
print('Confusion matrix (for Agglomerative): \n' ,C)
```

```
def cluster_class_mapping(predicted,true_labels):
    C = metrics.confusion_matrix(predicted,true_labels)
    mapping = list(np.argmax(C,axis=1)) #for each row (cluster) find the best␣,→class in the co
nfusion matrix
    mapped_kmeans_labels = [mapping[l] for l in predicted]
    C2 = metrics.confusion_matrix(mapped_kmeans_labels,true_labels)
    return mapped_kmeans_labels,C2


mapped_kmeans_labels,C = cluster_class_mapping(k_predicted,actual)
print('Confusion matrix (for k-means): \n' ,C)
```

```
********* Agglomerative Clustering *********
```

Percision:	Recall:	Confusion matrix (for Agglomerative):
0.9074233883276631	0.8857142857142857	[[60 784 191] [761 8 1] [0 12 563]]

```
********* K-means Clustering *********
```

Percision:	Recall:	Confusion matrix (for k-means):
0.9451027664328102	0.9411764705882353	[[770 7 0] [51 777 62] [0 20 693]]

```
****Logistic Regression mean scores****
```

```
mean score test accuracy: 0.930252100840336
```

```
mean score test precision weighted: 0.9311110649318362

mean score test recall weighted: 0.930252100840336

mean score test f1-measure weighted: 0.9303521992152968

Confusion Matrix: [[774  34  13]
                  [ 27 749  28]
                  [ 14  50 691]]
```

****SVM mean scores****

```
mean score test accuracy: 0.9315126050420167

mean score test precision weighted: 0.9318336338131683

mean score test recall weighted: 0.9315126050420167

mean score test f1-measure weighted: 0.9315318350559185

Confusion Matrix: [[781  28  12]
                  [ 26 741  37]
                  [ 14  46 695]]
```

****Decision Tree mean scores****

```
mean score test accuracy: 0.8176470588235294

mean score test precision weighted: 0.8192663317138272

mean score test recall weighted: 0.8176470588235294

mean score test f1-measure weighted: 0.818011699282092

Confusion Matrix: [[695  77  49]
                  [ 75 631  98]
                  [ 31 100 624]]
```

****k-NN mean scores****

```
mean score test accuracy: 0.799579831932773

mean score test precision weighted: 0.8017776971681257

mean score test recall weighted: 0.799579831932773

mean score test f1-measure weighted: 0.7994065502172664

Confusion Matrix: [[651  61 109]
                  [ 86 626  92]
                  [ 43  86 626]]
```

****Naive-Bayes mean scores****

```
mean score test accuracy: 0.911344537815126

mean score test precision weighted: 0.9159718744022938

mean score test recall weighted: 0.911344537815126

mean score test f1-measure weighted: 0.9119924610904245

Confusion Matrix: [[744  67  10]
                  [ 20 759  25]
                  [ 14  75 666]]
```

Clustering performs better in all cases as far as confusion-matrix is considered. However, for precision and recall SVM and Naive-Bayes scales better.

But...What is confusion matrix and what is the meaning of this matrix?

ASSIGNMENT 3

DATA MINING

Well, confusion matrix is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.

Predicted Values \ Actual Values	Positive (1)	Negative (0)
Positive (1)	TP	FP
Negative (0)	FN	TN

Well...That TP, FP, FN, TN what is all about?

TP stands for True Positive. That means you predicted positive and it's true.

FP stands for False Positive. You predicted positive and it's false.

FN stands for False Negative. You predicted negative and it's false.

TN stands for True Negative. You predicted negative and it's true.

So the conclusion of the confusion matrix is that we want our output to have as many "items" as possible to be in diagonal. For the perfect classifier the outputs are only in the diagonal.

Example: *We want to predict if it is a cat, a dog or a rabbit.*

	Actual Dog	Actual Cat	Actual Rabbit
Classified Dog	23	12	7
Classified Cat	11	29	13
Classified Rabbit	4	10	24