TELEGRAM CHATBOT WITH MACHINE LEARNING

Implementation of a Chatbot to work with Pizza orders. \ Built with Telegram API and use of 2 approches with Machine Learning models:

- To predict and suggest pizza toppings to clients;
- To analyze post-sale feedback and comments.

Models implemented:

- Random Forest;
- Logistic regression;
- Naive Bayes;
- KNN;
- Term Frequency-Inverse Document Frequency (TF-IDF);
- Bag of words (BOW).

The BOW and TF-IDF was made with a tweet file.

In this Jupyter file you will find the construction of the models.

In the .py file you can analyze the implementation of the Bot and use of the models built here.

Important: This project was built for a Brazilian client, so the features, \ objects, and models have their names in Portuguese.

In [ ]:

```
# Libraries

import numpy as np
from datetime import time
from datetime import datetime

from joblib import dump, load
import pickle

import csv
import pandas as pd

# Text processing
import re
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from unicodedata import normalize

# Classifiers
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier



# Model evaluation metrics
from sklearn.metrics import classification_report, accuracy_score, f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
```

Loading Data

In [ ]:

```
# Loading the tweet file
df = pd.read_csv('tweet_csv.csv', encoding='Latin1', sep =';')
print('Dimensao Arquivo:', df.shape)
df.head(5)
```

```
Dimensao Arquivo: (7322, 24)
```

Out[ ]:

	lin	Created At	Text	Geo Coordinates.latitude	Geo Coordinates.longitude	User Location
0	0	Sun Jan 08 01:22:05 +0000 2017	???? @ Catedral de Santo Antônio - Governador ...	NaN	NaN	Brasil
1	1	Sun Jan 08 01:49:01 +0000 2017	? @ Governador Valadares, Minas Gerais https:/...	-419.333	-18.85	NaN
2	2	Sun Jan 08 01:01:46 +0000 2017	?? @ Governador Valadares, Minas Gerais https:...	-419.333	-18.85	NaN
3	3	Wed Jan 04 21:43:51 +0000 2017	??? https://t.co/BnDsO34qK0	NaN	NaN	NaN
4	4	Mon Jan 09 15:08:21 +0000 2017	??? PSOL vai questionar aumento de vereadores ...	NaN	NaN	NaN

Removing unnecessary columns

In [ ]:

```
df = df[['Text', 'Classificacao']]
#df = df[df.Classificacao != 'Neutro']
df.head(5)
```

Out[ ]:

	Text	Classificacao
0	???? @ Catedral de Santo Antônio - Governador ...	Neutro
1	? @ Governador Valadares, Minas Gerais https:/...	Neutro
2	?? @ Governador Valadares, Minas Gerais https:...	Neutro
3	??? https://t.co/BnDsO34qK0	Neutro
4	??? PSOL vai questionar aumento de vereadores ...	Negativo

We created a customized stop words file.

In [ ]:

```
stop_words = list(pd.read_csv('stop_word.csv', encoding='Latin1', sep =';'))
stop_words[:10]
```

Out[ ]:

```
['de', 'a', 'o', 'que', 'e', 'do', 'da', 'em', 'um', 'para']
```

We also created a function to clear our text, facilitating future applications

In [ ]:

```
stop_words = set(stop_words) # creates list of words to be removed

def clean_text(text):
    text = text.lower() # Leaves all lowercase characters
    text = normalize('NFKD', text).encode('ASCII', 'ignore').decode('ASCII')
    text = [re.sub(r'^https?:\/\/.[\r\n]', ' ', word) for word in text.split(" ")]
    text = " ".join(text) # Join text again separated by space
    text = re.sub(r'<[^>]+>',' ',text) # Change the '<br />' section to space
    text = re.sub(r'[^a-z]', ' ', text) # Changes scores and numbers to space

    # Change single characters to space
    text = re.sub(r"\s+[a-z]\s+", ' ', text)

    # Removes repeated characters over 2 times
    text = re.sub(r'([a-z])(?=\1{2,})', "", text)
    text = re.sub(r'[ ]{1,}',' ',text) # Removes duplicate spaces in the string
    text = text.strip() # Remove spaces at the beginning and end of the string

    # Removes words found in the stop_sword list
    #text = [word for word in text.split(" ") if not word in stop_words]
    #text = " ".join(text) #Une o texto novamente separado por espaco
    return text
```

Applying cleanliness to training reviews

In [ ]:

```
example = "MARIAAAAAAAS ESTAVA CORRendU LOUCAMENTI NA AVENIDA, QUANDO TROPÇÇÕU123 http
s://baçblax54G"
print("Exemplo Antes :", example)
print("Exemplo Depois :", clean_text(example))
```

```
Exemplo Antes : MARIAAAAAAAS ESTAVA CORRendU LOUCAMENTI NA AVENIDA, QUANDO
TROPÇÇÕU123 https://baçblax54G
Exemplo Depois : mariaas estava correndu loucamenti na avenida quando trop
ccou https bacblax g
```

In [ ]:

```
df['TEXTO_LIMPO'] = df.Text.apply(lambda x: clean_text(x))
print('Dimensao Treino:', df.shape)
df.head(5)
```

```
Dimensao Treino: (7322, 3)
```

Out[ ]:

	Text	Classificacao	TEXTO_LIMPO
0	???? @ Catedral de Santo Antônio - Governador ...	Neutro	catedral de santo antonio governador valadares...
1	? @ Governador Valadares, Minas Gerais https:/...	Neutro	governador valadares minas gerais https co thi...
2	?? @ Governador Valadares, Minas Gerais https:...	Neutro	governador valadares minas gerais https co dpk...
3	??? https://t.co/BnDsO34qK0	Neutro	https co bndso qk
4	??? PSOL vai questionar aumento de vereadores ...	Negativo	psol vai questionar aumento de vereadores pref...

APPLICATION OF THE MODEL IN THE TEST FILE

In [ ]:

```
train, valid = train_test_split(df, test_size = 0.3, random_state=13)
```

Now for the benchmark we will create and apply Logistic Regression, Naive Bayes \ and Random Forest models, using bag of words (bow) and also \ Term Frequency-Inverse Document Frequency (tf-idf).

In [ ]:

```
bow = CountVectorizer()
tfidf = TfidfVectorizer()

x_train_bow = bow.fit_transform(train['TEXTO_LIMPO'])
x_train_tfidf = tfidf.fit_transform(train['TEXTO_LIMPO'])

x_valid_bow = bow.transform(valid['TEXTO_LIMPO'])
x_valid_tfidf = tfidf.transform(valid['TEXTO_LIMPO'])
```

Training of Logistic Regression, Naive Bayes and Random Forest models with bow and tfidf without manipulation of parameters and hyperparameters.

In [ ]:

```
# Logistic Regression
model_lr_bow = LogisticRegression()
model_lr_tfidf = LogisticRegression()

model_lr_bow.fit(x_train_bow, train['Classificacao'])
model_lr_tfidf.fit(x_train_tfidf, train['Classificacao'])

# Naive Bayes
model_nb_bow = MultinomialNB()
model_nb_tfidf = MultinomialNB()

model_nb_bow.fit(x_train_bow, train['Classificacao'])
model_nb_tfidf.fit(x_train_tfidf, train['Classificacao'])

# Random Forest
model_rf_bow = RandomForestClassifier()
model_rf_tfidf = RandomForestClassifier()

model_rf_bow.fit(x_train_bow, train['Classificacao'])
model_rf_tfidf.fit(x_train_tfidf, train['Classificacao'])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:9
40: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Measuring model results in the validation file.

In [ ]:

```
# Logistic Regression
y_pred_bow = model_lr_bow.predict(x_valid_bow)
y_pred_tfidf = model_lr_tfidf.predict(x_valid_tfidf)

print("Valid LR BOW Accuracy:", accuracy_score(valid['Classificacao'],\
                                              y_pred_bow))
print('Valid LR BOW Confusion matrix:')
print(confusion_matrix(y_pred_bow, valid['Classificacao']))


# We use the 'weighted' parameter as it does the calculation for each label
# and finds its support weighted average (the number of true instances for
# each label). It already considers the imbalance of the label.
print('F1-score LR BOW:',f1_score(valid['Classificacao'], y_pred_bow,\
                                  average='weighted'))

print("\nValid LR TFIDF Accuracy:", accuracy_score(valid['Classificacao'],\
                                                  y_pred_tfidf))
print('Valid LR TFIDF Confusion matrix:')
print(confusion_matrix(y_pred_tfidf, valid['Classificacao']))
print('F1-score LR TFIDF:',f1_score(valid['Classificacao'], y_pred_tfidf,\
                                    average='weighted'))



# Naive Bayes
y_pred_bow = model_nb_bow.predict(x_valid_bow)
y_pred_tfidf = model_nb_tfidf.predict(x_valid_tfidf)

print("\nValid NB BOW Accuracy:", accuracy_score(valid['Classificacao'],\
                                                y_pred_bow))
print('Valid NB BOW Confusion matrix:')
print(confusion_matrix(y_pred_bow, valid['Classificacao']))
print('F1-score NB BOW:',f1_score(valid['Classificacao'], y_pred_bow,\
                                  average='weighted'))

print("\nValid NB TFIDF Accuracy:", accuracy_score(valid['Classificacao'],\
                                                  y_pred_tfidf))
print('Valid NB TFIDF Confusion matrix:')
print(confusion_matrix(y_pred_tfidf, valid['Classificacao']))
print('F1-score NB TFIDF:',f1_score(valid['Classificacao'], y_pred_tfidf,\
                                    average='weighted'))



# Random Forest
y_pred_bow = model_rf_bow.predict(x_valid_bow)
y_pred_tfidf = model_rf_tfidf.predict(x_valid_tfidf)

print("\nValid RF BOW Accuracy:", accuracy_score(valid['Classificacao'],\
                                                y_pred_bow))
print('Valid RF BOW Confusion matrix:')
print(confusion_matrix(y_pred_bow, valid['Classificacao']))
print('F1-score RF BOW:',f1_score(valid['Classificacao'], y_pred_bow,\
                                  average='weighted'))

print("\nValid RF TFIDF Accuracy:", accuracy_score(valid['Classificacao'],\
                                                  y_pred_tfidf))
print('Valid RF TFIDF Confusion matrix:')
print(confusion_matrix(y_pred_tfidf, valid['Classificacao']))
```

```python
print('F1-score RF TFIDF:',f1_score(valid['Classificacao'], y_pred_tfidf,\
                                    average='weighted'))
```

```
Valid LR BOW Accuracy: 0.9649522075557578
Valid LR BOW Confusion matrix:
[[538  18   1]
 [ 21 692  23]
 [  4  10 890]]
F1-score LR BOW: 0.965031059081339

Valid LR TFIDF Accuracy: 0.9522075557578517
Valid LR TFIDF Confusion matrix:
[[531  24   1]
 [ 29 670  22]
 [  3  26 891]]
F1-score LR TFIDF: 0.9521810746032993

Valid NB BOW Accuracy: 0.9390077378243059
Valid NB BOW Confusion matrix:
[[535  42   3]
 [ 23 641  24]
 [  5  37 887]]
F1-score NB BOW: 0.9387309256468498

Valid NB TFIDF Accuracy: 0.9280837505689576
Valid NB TFIDF Confusion matrix:
[[534  43   1]
 [ 21 608  16]
 [  8  69 897]]
F1-score NB TFIDF: 0.9271188346463523

Valid RF BOW Accuracy: 0.9672280382339554
Valid RF BOW Confusion matrix:
[[535  12   1]
 [ 24 701  24]
 [  4   7 889]]
F1-score RF BOW: 0.9673398439000127

Valid RF TFIDF Accuracy: 0.9617660446062813
Valid RF TFIDF Confusion matrix:
[[535  16   1]
 [ 25 690  25]
 [  3  14 888]]
F1-score RF TFIDF: 0.9618659968051969
```

We chose the Random Forest models with BOW due to their accuracy and F1-score

ML Model for pizza orders

In [ ]:

```
df_p = pd.read_excel("pedidos.xlsx") # Importing the historical data
type(df_p) # Verification
df_p.head()
```

Out[ ]:

	ID_TELEFONE	GENERO	IDADE	CEP	HORA	TIPO	PEDIDO
0	1	HOMEM	28	17050250	18:15:00	SALGADA	A MODA DO CHEFE
1	2	HOMEM	28	17051500	18:15:00	SALGADA	VEGETARIANA
2	3	MULHER	19	17053013	18:44:00	SALGADA	CALABRESA
3	4	MULHER	19	17056035	18:44:00	SALGADA	CALABRESA
4	5	MULHER	28	17010160	18:44:00	SALGADA	RÚCULA C/ TOMATE SECO

In [ ]:

```
# Checking the types of each column

df_p.dtypes
```

Out[ ]:

```
ID_TELEFONE     int64
GENERO         object
IDADE           int64
CEP             int64
HORA           object
TIPO           object
PEDIDO         object
dtype: object
```

In [ ]:

```
# Checking GENERO variable distribution and type
df_p.GENERO.value_counts()
```

Out[ ]:

```
MULHER          236
HOMEM           199
NÃO_INFORMAR      1
Name: GENERO, dtype: int64
```

In [ ]:

```
# Changing variable type to categorical
df_p['GENERO'] = df_p['GENERO'].astype('category')
type(df_p.GENERO)
```

Out[ ]:

```
pandas.core.series.Series
```

In [ ]:

```python
# Here we create age ranges
df_p.loc[df_p.IDADE <= 23, 'FAIXA_IDADE'] = 'ATE_23_ANOS'
df_p.loc[(df_p.IDADE > 23) & (df_p.IDADE <= 30), 'FAIXA_IDADE'] = 'ATE_30_ANOS'
df_p.loc[(df_p.IDADE > 30) & (df_p.IDADE <= 40), 'FAIXA_IDADE'] = 'ATE_40_ANOS'
df_p.loc[(df_p.IDADE > 40) & (df_p.IDADE <= 50), 'FAIXA_IDADE'] = 'ATE_50_ANOS'
df_p.loc[(df_p.IDADE > 50) & (df_p.IDADE <= 70), 'FAIXA_IDADE'] = 'ATE_70_ANOS'
df_p.loc[df_p.IDADE > 70, 'FAIXA_IDADE'] = 'ACIMA_70_ANOS'

# Changing variable type to categorical
df_p['FAIXA_IDADE'] = df_p['FAIXA_IDADE'].astype('category')
df_p.head()
```

Out[ ]:

	ID_TELEFONE	GENERO	IDADE	CEP	HORA	TIPO	PEDIDO	FAIXA_ID
0	1	HOMEM	28	17050250	18:15:00	SALGADA	A MODA DO CHEFE	ATE_30_A
1	2	HOMEM	28	17051500	18:15:00	SALGADA	VEGETARIANA	ATE_30_A
2	3	MULHER	19	17053013	18:44:00	SALGADA	CALABRESA	ATE_23_A
3	4	MULHER	19	17056035	18:44:00	SALGADA	CALABRESA	ATE_23_A
4	5	MULHER	28	17010160	18:44:00	SALGADA	RÚCULA C/ TOMATE SECO	ATE_30_A

In [ ]:

```python
# Time column adjustment for datetime

df_p['HORA'] = df_p['HORA'].apply(lambda x: datetime.strptime(x, '%H:%M:%S'))
df_p.head()
```

Out[ ]:

	ID_TELEFONE	GENERO	IDADE	CEP	HORA	TIPO	PEDIDO	FAIXA_ID
0	1	HOMEM	28	17050250	1900-01-01 18:15:00	SALGADA	A MODA DO CHEFE	ATE_30_A
1	2	HOMEM	28	17051500	1900-01-01 18:15:00	SALGADA	VEGETARIANA	ATE_30_A
2	3	MULHER	19	17053013	1900-01-01 18:44:00	SALGADA	CALABRESA	ATE_23_A
3	4	MULHER	19	17056035	1900-01-01 18:44:00	SALGADA	CALABRESA	ATE_23_A
4	5	MULHER	28	17010160	1900-01-01 18:44:00	SALGADA	RÚCULA C/ TOMATE SECO	ATE_30_A

In [ ]:

```python
# Extracting the hour values

df_p['HORA_OK'] = df_p['HORA'].dt.hour
df_p['MINUTO_OK'] = df_p['HORA'].dt.minute
df_p = df_p.drop(columns=['HORA'])
df_p.head()
```

Out[ ]:

	ID_TELEFONE	GENERO	IDADE	CEP	TIPO	PEDIDO	FAIXA_IDADE	HO
0	1	HOMEM	28	17050250	SALGADA	A MODA DO CHEFE	ATE_30_ANOS	
1	2	HOMEM	28	17051500	SALGADA	VEGETARIANA	ATE_30_ANOS	
2	3	MULHER	19	17053013	SALGADA	CALABRESA	ATE_23_ANOS	
3	4	MULHER	19	17056035	SALGADA	CALABRESA	ATE_23_ANOS	
4	5	MULHER	28	17010160	SALGADA	RÚCULA C/ TOMATE SECO	ATE_30_ANOS	

In [ ]:

```python
# Creating groups for the minutes

df_p.loc[df_p.MINUTO_OK <= 30, 'FAIXA_MINUTO'] = 'ATE_30_MINUTOS'
df_p.loc[(df_p.MINUTO_OK > 30), 'FAIXA_MINUTO'] = 'ACIMA_30_MINUTOS'

# Changing variable type to categorical
df_p['FAIXA_MINUTO'] = df_p['FAIXA_MINUTO'].astype('category')
df_p = df_p.drop(columns=['MINUTO_OK'])
df_p.head()
```

Out[ ]:

	ID_TELEFONE	GENERO	IDADE	CEP	TIPO	PEDIDO	FAIXA_IDADE	HO
0	1	HOMEM	28	17050250	SALGADA	A MODA DO CHEFE	ATE_30_ANOS	
1	2	HOMEM	28	17051500	SALGADA	VEGETARIANA	ATE_30_ANOS	
2	3	MULHER	19	17053013	SALGADA	CALABRESA	ATE_23_ANOS	
3	4	MULHER	19	17056035	SALGADA	CALABRESA	ATE_23_ANOS	
4	5	MULHER	28	17010160	SALGADA	RÚCULA C/ TOMATE SECO	ATE_30_ANOS	

In [ ]:

```
# Target

# Changing variable type to categorical
df_p['PEDIDO'] = df_p['PEDIDO'].astype('category')
type(df_p.PEDIDO)
```

Out[ ]:

```
pandas.core.series.Series
```

In [ ]:

```
# Removing columns that will not be used

df_p = df_p.drop(columns=['ID_TELEFONE','IDADE', 'TIPO'])
```

In [ ]:

```
# Creating dummy for categorical variables

df_p_Dummies_gender = pd.get_dummies(df_p['GENERO'], prefix = 'category')

df_p_Dummies_age = pd.get_dummies(df_p['FAIXA_IDADE'], prefix = 'category')

df_p_Dummies_minute = pd.get_dummies(df_p['FAIXA_MINUTO'], prefix = 'category')
df_p_Dummies_minute.head()
```

Out[ ]:

	category_ACIMA_30_MINUTOS	category_ATE_30_MINUTOS
0	0	1
1	0	1
2	1	0
3	1	0
4	1	0

In [ ]:

```
# Concatenating the dummies and removing the original columns

df_p = pd.concat([df_p, df_p_Dummies_gender, df_p_Dummies_age,\
                  df_p_Dummies_minute], axis=1)

df_p = df_p.drop(columns=['GENERO','FAIXA_IDADE','FAIXA_MINUTO'])
df_p.head()
```

Out[ ]:

	CEP	PEDIDO	HORA_OK	category_HOMEM	category_MULHER	category_NÃO
0	17050250	A MODA DO CHEFE	18	1	0	
1	17051500	VEGETARIANA	18	1	0	
2	17053013	CALABRESA	18	0	1	
3	17056035	CALABRESA	18	0	1	
4	17010160	RÚCULA C/ TOMATE SECO	18	0	1	

In [ ]:

```
# Separating training and testing sets

var = np.random.rand(len(df_p)) < 0.8

treino_p = df_p[var]

teste_p = df_p[~var]
```

In [ ]:

```
# Separation of dependent variables from target

x_treino_p = treino_p.drop(columns=['PEDIDO'])
y_treino_p = treino_p.loc[:,['PEDIDO']]

x_teste_p = teste_p.drop(columns=['PEDIDO'])
y_teste_p = teste_p.loc[:,['PEDIDO']]
```

In [ ]:

```
# Applying the KNN model and fitting to the file

classifier = KNeighborsClassifier(n_neighbors=19)
classifier.fit(x_treino_p, y_treino_p)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConver
sionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing import
s until
```

Out[ ]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=19, p=2,
                     weights='uniform')
```

In [ ]:

```
# Predict in the file and already adding the column to the end

x_treino_p['pred_y_knn'] = classifier.predict(x_treino_p)
x_treino_p.head()
```

Out[ ]:

	CEP	HORA_OK	category_HOMEM	category_MULHER	category_NÃO_INFORMAR	ca
0	17050250	18	1	0	0	
1	17051500	18	1	0	0	
2	17053013	18	0	1	0	
3	17056035	18	0	1	0	
4	17010160	18	0	1	0	

In [ ]:

```
# Returning the original label and doing cross-validation

treino_p = pd.concat([x_treino_p, y_treino_p], axis=1)

pd.crosstab(treino_p["PEDIDO"],treino_p["pred_y_knn"]\
            ).apply(lambda r: r/r.sum()*100, axis=1)

# As we can analyze in the table below, the model did not achieve good
# performance even on the training set, as it could not predict all the
# labels. So, we will not apply to the test set. This phenomenon is mainly
# linked to the volume of samples we are able to collect.
```

Out[ ]:

pred_y_knn PEDIDO	A MODA DA CASA	CALABRESA	FRANGO C/ CREAM CHEESE	FRANGO C/ MILHO	MARGUERITA	PORTUGUESA
A MODA DA CASA	4.545455	59.090909	4.545455	4.545455	0.000000	27.272727
A MODA DO CHEFE	0.000000	0.000000	0.000000	0.000000	0.000000	100.000000
ABOBRINHA	0.000000	0.000000	0.000000	0.000000	0.000000	50.000000
ALICHE	0.000000	33.333333	0.000000	0.000000	0.000000	66.666667
ATUM	0.000000	20.000000	20.000000	20.000000	0.000000	20.000000
CALABRESA	2.380952	60.714286	2.380952	4.761905	1.190476	27.380952
CARNE SECA	0.000000	40.000000	0.000000	0.000000	0.000000	60.000000
FRANGO C/ CREAM CHEESE	2.325581	53.488372	9.302326	0.000000	0.000000	30.232558
FRANGO C/ MILHO	0.000000	42.857143	4.761905	33.333333	0.000000	19.047619
MARGARIDA	0.000000	100.000000	0.000000	0.000000	0.000000	0.000000
MARGUERITA	0.000000	46.428571	0.000000	10.714286	10.714286	25.000000
MILHO COM CATUPIRY	0.000000	62.500000	0.000000	0.000000	0.000000	37.500000
MUSSARELA	0.000000	76.923077	0.000000	7.692308	0.000000	15.384615
PALMITO ESPECIAL	0.000000	37.500000	12.500000	0.000000	12.500000	37.500000
PORTUGUESA	0.000000	47.500000	0.000000	6.250000	0.000000	43.750000
RÚCULA C/ TOMATE SECO	0.000000	55.555556	0.000000	11.111111	0.000000	33.333333
SICILIANA	25.000000	50.000000	0.000000	0.000000	0.000000	0.000000
VEGETARIANA	0.000000	44.444444	0.000000	0.000000	0.000000	38.888889

In [ ]:

```
# Applying the Random Forest model and fitting to the file

# to remove the predictor from the previous model.
x_treino_p = x_treino_p.drop(columns=['pred_y_knn'])

model_rf = RandomForestClassifier()
model_rf.fit(x_treino_p, y_treino_p['PEDIDO'])
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
# Predict in the file and already adding the column to the end

x_treino_p['pred_y_rf'] = model_rf.predict(x_treino_p)
x_treino_p.head()
```

Out[ ]:

	CEP	HORA_OK	category_HOMEM	category_MULHER	category_NÃO_INFORMAR	ca
0	17050250	18	1	0	0	
1	17051500	18	1	0	0	
2	17053013	18	0	1	0	
3	17056035	18	0	1	0	
4	17010160	18	0	1	0	

In [ ]:

```python
# Returning the original label and doing cross-validation

treino_p = pd.concat([x_treino_p, y_treino_p], axis=1)

pd.crosstab(treino_p["PEDIDO"],treino_p["pred_y_rf"]\
            ).apply(lambda r: r/r.sum()*100, axis=1)
```

Out[ ]:

pred_y_rf	A MODA DA CASA	A MODA DO CHEFE	ABOBRINHA	ALICHE	ATUM	CALABRESA	CARNE SECA	FI (C
PEDIDO								
A MODA DA CASA	100.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.
A MODA DO CHEFE	0.000000	100.0	0.0	0.0	0.0	0.0	0.0	0.
ABOBRINHA	0.000000	0.0	100.0	0.0	0.0	0.0	0.0	0.
ALICHE	0.000000	0.0	0.0	100.0	0.0	0.0	0.0	0.
ATUM	0.000000	0.0	0.0	0.0	100.0	0.0	0.0	0.
CALABRESA	0.000000	0.0	0.0	0.0	0.0	100.0	0.0	0.
CARNE SECA	0.000000	0.0	0.0	0.0	0.0	0.0	100.0	0.
FRANGO C/ CREAM CHEESE	2.325581	0.0	0.0	0.0	0.0	0.0	0.0	97.
FRANGO C/ MILHO	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.
MARGARIDA	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.
MARGUERITA	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.
MILHO COM CATUPIRY	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.
MUSSARELA	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.
PALMITO ESPECIAL	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.
PORTUGUESA	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.
RÚCULA C/ TOMATE SECO	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.
SICILIANA	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.
VEGETARIANA	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.

In [ ]:

```
# Predict in the test set

x_teste_p['pred_y_rf'] = model_rf.predict(x_teste_p)
x_teste_p.head()
```

Out[ ]:

	CEP	HORA_OK	category_HOMEM	category_MULHER	category_NÃO_INFORMAR	c
13	17550000	18	0	1	0	
24	17014000	19	1	0	0	
27	17025300	19	0	1	0	
28	17055250	19	0	1	0	
29	17064450	19	1	0	0	

In [ ]:

```
# Returning the original label and doing cross-validation

teste_p = pd.concat([x_teste_p, y_teste_p], axis=1)

pd.crosstab(teste_p["PEDIDO"],teste_p["pred_y_rf"]\
            ).apply(lambda r: r/r.sum()*100, axis=1)

# The random forest model also did not achieve a satisfactory result and as
# mentioned in the knn model, the main factor is the volume of samples that
# we were able to collect. The RF model is implemented in the chatbot .py file,
# only for the purpose of conceptual until a larger volume of samples are
# collected, when the project is in production.
```

Out[ ]:

pred_y_rf	A MODA DA CASA	CALABRESA	FRANGO C/ CREAM CHEESE	FRANGO C/ MILHO	MARGUERITA	MILHO COM CATUPIRY
PEDIDO						
A MODA DA CASA	0.000000	75.000000	0.000000	0.000000	0.000000	0.000000
A MODA DO CHEFE	0.000000	0.000000	0.000000	50.000000	0.000000	0.000000
CALABRESA	11.111111	16.666667	16.666667	0.000000	16.666667	5.555556
CARNE SECA	0.000000	100.000000	0.000000	0.000000	0.000000	0.000000
ESPANHOLA	0.000000	100.000000	0.000000	0.000000	0.000000	0.000000
FRANGO C/ CREAM CHEESE	27.272727	18.181818	9.090909	0.000000	0.000000	0.000000
FRANGO C/ MILHO	0.000000	100.000000	0.000000	0.000000	0.000000	0.000000
MARGUERITA	0.000000	30.000000	20.000000	10.000000	20.000000	0.000000
MILHO COM CATUPIRY	0.000000	0.000000	100.000000	0.000000	0.000000	0.000000
MUSSARELA	0.000000	16.666667	0.000000	16.666667	16.666667	0.000000
PALMITO ESPECIAL	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
PORTUGUESA	0.000000	8.333333	8.333333	8.333333	0.000000	0.000000
ROMEU E JULIETA	0.000000	0.000000	0.000000	100.000000	0.000000	0.000000
RÚCULA C/ TOMATE SECO	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
SICILIANA	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
VEGETARIANA	0.000000	0.000000	0.000000	0.000000	50.000000	0.000000

In [ ]:

```
# F1-Score Random Forest

print('F1-score RANDOM FOREST PIZZA:',f1_score(teste_p["PEDIDO"],\
                                               teste_p["pred_y_rf"],\
                                               average='weighted'))
```

```
F1-score RANDOM FOREST PIZZA: 0.1291289038437998
```

In [ ]:

```
with open('preditor_pizza.pkl', 'wb') as f:
  pickle.dump(model_rf, f)
```

In [ ]:

```
with open('preditor_avaliacao_bow.pkl', 'wb') as f:
  pickle.dump(model_rf_bow, f)
```

In [ ]:

```
with open('bow.pkl', 'wb') as f:
  pickle.dump(bow, f)
```