MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

CNG 562
MACHINE LEARNING

PROJECT

Report

Nisa Nur Odabaş
Kaan Taha Köken

June 2, 2020

Contents

List of Figures

1 Abstract

For this project, we examined recipes from around the world. We were really interested to see if we could learn something about the relationships of different cuisines throughout the world. In order to explore this topic, we chose to use recipe data. In particular, we used the list of ingredients for almost 40 thousands different recipes and ran several machine learning models. This project provides an overview of our process for using NLP methods to process the data, run different types of machine learning methods, experimenting different types of tuning, boosting and finally some results obtained from different models.

2 Introduction

In this project, we wanted to work with a data set that we did not use before. We looked at the challenges in the Kaggle to decide what to use. We decided to go with the challenge called What's Cooking?. As we can understand the name of the challenge, it is related to foods. After a quick look at the challenge, we will make a prediction of food origin. The dataset that we took, basically, consists of the cuisine of the food and its recipe, and in the end, we are going to make predictions on cuisine label.

What makes more this dataset interesting is in this dataset, the number of features is found different in each row.

Throughout the project, we have examined different techniques of preprocessing, different types of Machine Learning models. Also, to make our data more meaningful, we have used feature engineering. To improve our result, we have also used parameter tuning and boosting techniques.

3 Problem Statement

Our dataset is called Recipe Ingredients Dataset which is provided by Yummly. Since we found it from a competition in the Kaggle, there is a test set that does not have the cuisine labels. Therefore, we decided to use their train set as train and test sets.

Another problem in this dataset is that since every recipe has different number of ingredients, the number of features are not same. Since the number of feature size is not certain, estimating the cuisine labels will be challenging.

4 Problem Analysis

The dataset is a json file which consists id, cuisine and ingredients columns. It has 20 classes as you can see in figure-1. It also has only one feature which is ingredients. The challenge is that features' sizes are not equal since every meal has different number of ingredients. Moreover, the dataset consists of 39774 samples in total.

```
['greek' 'southern_us' 'filipino' 'indian' 'jamaican' 'spanish' 'italian'
 'mexican' 'chinese' 'british' 'thai' 'vietnamese' 'cajun_creole'
 'brazilian' 'french' 'japanese' 'irish' 'korean' 'moroccan' 'russian']
```

Figure 1: Classes / Cuisines

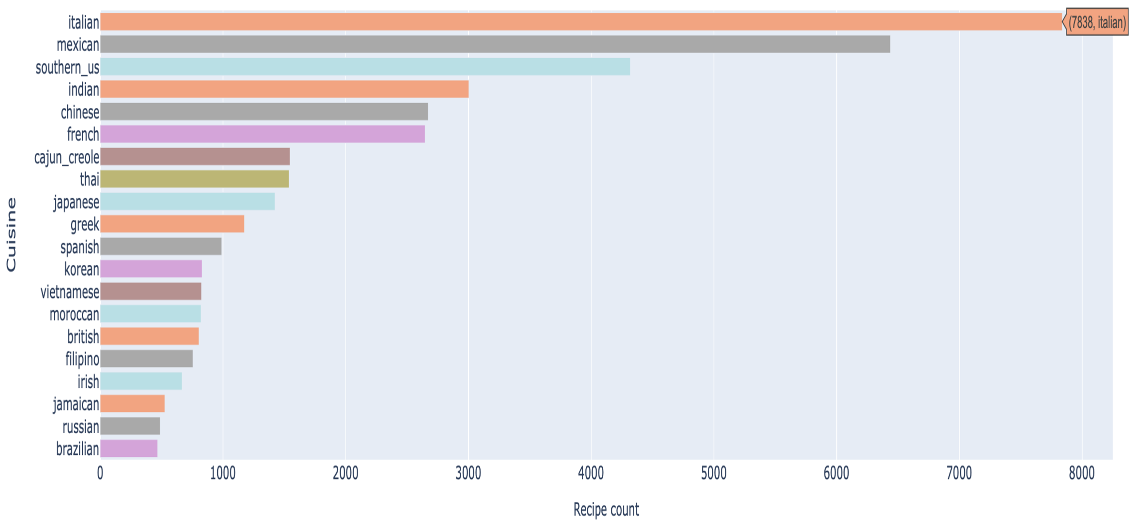

Figure 2: Count of recipes per cuisine

5 Approach

First of all, since we have separate train and test sets, but we wanted to see accuracy on the model, we have split our train data as **%70** train **%30** test to validate our models and see how they are doing. Then, we applied preprocessing on our dataset.

Figure 3: Train/Test Split percentage

5.1 Preprocessing

First of all, we checked that if we have a missing value in columns, and we found out that there is no missing data(4).

```
              # of Rows, Columns: (39774, 3)
              Total missing  Percent missing
ingredients               0              0.0
cuisine                   0              0.0
id                        0              0.0
```

Figure 4: Missing values

Since we were dealing with texts, we needed to do some corrections. The first step was converting all ingredients into lowercase. Then, we removed '-' and spaces in some ingredients such as 'low-fat' and 'fish sauce'. We also did lemmatization to convert some words into its base form. We used **WordNetLemmatizer** which is provided by **NLTK (Natural Language Toolkit)**.

After doing some corrections in data, we tried different feature engineering techniques. Firstly, we tried term frequency-inverse document frequency (TF–IDF). It basically weights the word counts by a measure of how often they appear in the recipes. In order to achieve this, we used **TfidfVectorizer** from Scikit-Learn. Another approach that we tried for vectorization is also based on word count. However, this time, we directly counted number of times each ingredient appears. We used **CountVectorizer** from Scikit-Learn for this approach. Finally, we also tried hashing to vectorize data. **HashingVectorizer** from Scikit-Learn uses the hashing trick to find the token string name to feature integer index mapping.

We also needed to do some changes in our classes since they are string type. In order to use these labels, we encoded them. Then, when we wanted to see predicted labels, we used label decoder.

5.2 Model

This project is on ***Natural Language Processing*** in short **NLP**, and we wanted to use models that suit NLP problems. For the purpose of that, we went over various classifiers, and we chose a few of them that we thought suits best to our problem. As a final thought, we decided to try **Support Vector Machine**, **Linear Support Vector Machine**, **Random Forest**, **Logistic Regression**, **KMeans**, **Multinomial Naive Bayes**.

After we chose our classifiers, we wanted try the base versions, and looked how they were doing. We tried our six models with three different datasets that converted with feature engineering tools.

```
x_train_tfid = tfid_vectorizer.fit_transform(train['x'].values)
x_train_tfid.sort_indices()
x_test_tfid = tfid_vectorizer.transform(test['x'].values)

x_train_counter = counter_vectorizer.fit_transform(train['x'].values)
x_train_counter.sort_indices()
x_test_counter = counter_vectorizer.transform(test['x'].values)

x_train_hash = hash_vectorizer.fit_transform(train['x'].values)
x_train_hash.sort_indices()
x_test_hash = hash_vectorizer.transform(test['x'].values)
```

Figure 5: Converting dataset

On our tests, we got good results on SVM, Linear SVM, Random Forest and Linear Regression. These models perform well and quite similarly on those three datasets. However, dataset that converted with **frequency-inverse document frequency (TF–IDF)** is giving best accuracies so far.

```
LR: 0.785538 (0.007259)
LSVM: 0.792477 (0.006236)
RF: 0.753885 (0.005074)
```

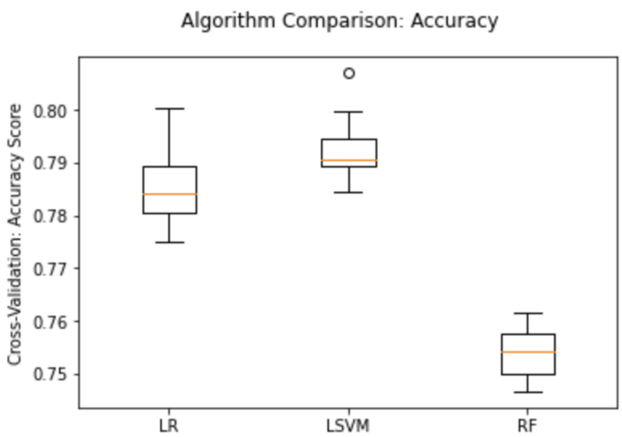

Figure 6: Model comparision 1

```
SVM [0.79886864 0.80817096 0.80125707 0.79748586 0.80575811]
Random Forest [0.74758014 0.75675676 0.74644877 0.74494029 0.75496605]
Kmeans [0.02778127 0.03293526 0.08736644 0.01181647 0.01106362]
```

Figure 7: Model comparision 2

After we got the results, we decided not to use **KMeans**, **Multinomial Naive Bayes** since we got very low accuracies. We continued on with the remaining four models. To achieve the best and most optimized of these models, we used Hyper Parameter Tuning techniques. Grid search can be thought of as an exhaustive search for selecting a model. We sets up a grid of hyperparameter values and for each combination, trains a model and scores on the testing data. On the otherhand, random search sets up a grid of hyperparameter values and selects random combinations to train the model and score. This allows you to explicitly control the number of parameter combinations that are attempted. The number of search iterations is set based on time or resources.

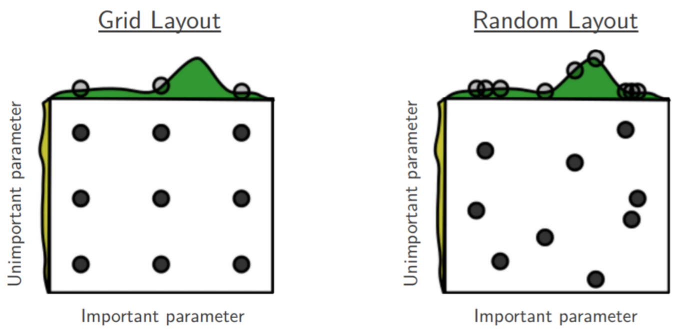

Figure 8: Grid Search vs Random Search

We tried to find best parameters for logistic regression using **GridSearch**. We saw that accuracy is generally around %78.

```
Logistic Regression Accuracy

penalty: l1,  C: 1   = 77.3823

penalty: l1,  C: 5   = 77.9533

penalty: l1,  C: 10  = 77.2170

penalty: l2,  C: 1   = 77.0913

penalty: l2,  C: 5   = 78.6070

penalty: l2,  C: 10  = 78.4059
```

Figure 9: Grid Search on Logistic Regression

On random forest, we have tried **GridSearch** and **RandomSearch** that Scikit-Learn provided, and the rest of the models we have used just GridSearch.

```
SVM
C: 0.1, Gamma: 1, Kernel: rbf
[0.6455378  0.64206178 0.64493534 0.64619253 0.64619253]
C: 0.1, Gamma: 1, Kernel: poly
[0.4578919  0.45617816 0.45114943 0.45456178 0.46354167]
C: 0.1, Gamma: 1, Kernel: linear
[0.71107919 0.71623563 0.71372126 0.71156609 0.71713362]
C: 0.1, Gamma: 0.1, Kernel: rbf
[0.56203986 0.55118534 0.56178161 0.55567529 0.55675287]
C: 0.1, Gamma: 0.1, Kernel: poly
[0.19716287 0.19701868 0.19701868 0.19701868 0.19701868]
C: 0.1, Gamma: 0.1, Kernel: linear
[0.71107919 0.71623563 0.71372126 0.71156609 0.71713362]
C: 0.1, Gamma: 0.01, Kernel: rbf
[0.20506375 0.20510057 0.20563937 0.20204741 0.20420259]
C: 0.1, Gamma: 0.01, Kernel: poly
```

Figure 10: Grid Search on SVM

```
Linear SVM
C: 0.1
[0.77787754 0.78017241 0.77496408 0.77047414 0.76849856]
C: 1
[0.78236667 0.7889727  0.78807471 0.77999282 0.77514368]
C: 10
[0.76028012 0.76742098 0.76939655 0.76167385 0.75520833]
C: 100
[0.7396301  0.74353448 0.73940374 0.73832615 0.72611351]
```

Figure 11: Grid Search on Linear SVM

As you can see above, when some parameters are decreased, our accuracy result are decreasing as well. After we got our tuned models, we looked their accuracy results. Tuning was improved a bit, but it did not improve much. We got our best result with SVM model.

In order to take the results a step further, we sent our optimized models into a method called **OneVsRest** in the Scikit-Learn library. OneVsRest is basically a heuristic method for using binary classification algorithms for multi-class classification. It involves splitting the multi-class dataset into multiple binary classification problems. A binary classifier is then trained on each binary classification problem and predictions are made using the model that is the most confident. We achieved that almost **82** percent accuracy.

Lastly, on the final model, we tried boosting methods to see if we were doing some improvements using our model. After we run **Adaboost** and **GradientBoost**, we did not get much improvements; therefore, we continued what we already had.

5.3 Final Model

We decided go with SVM model. For the regulation parameter C as **250**, for kernel as **rbf**', for the degree as **3**, for gamma **1.4**, and lastly we increased the cache size. As we discussed above, we used **frequency-inverse document frequency (TF–IDF)** normalized data.

For the four error state, we splitted our data into **%60** train, **%10** train-development, **%15** development, **%15** test.

```
Train Error,   e1:  0.8075433240999066

Train-Train Dev,   e2: 0.8075936377629553

Train-Dev,   e3 0.8082467314783774

Train-Test,   e4:  0.8052622758505111
```

Figure 12: Four Error Analysis

As you can see above, we got pretty stable result. Therefore, we can safely say that our model did solid job.

6 Conclusion

To sum up, we have experienced a unique dataset which we have not seen in the assignments before. Also, we had chance to explore dataset which has text features. We used many techniques we learned during this lesson such as featuring engineering and several estimators. We learned how to handle an NLP issue. As we mention above, we took this dataset from a competition in the Kaggle. Winner project has %82 accuracy. Therefore, we believe that we did a good solid job.

7 Appendix

7.1 Project Link

https://github.com/nisaodabas/CNG562-Project

7.2 Code

```
import numpy as np
import pandas as pd
import re
from pandas import read_csv
import matplotlib.pyplot as plt
import plotly.graph_objs as go
from plotly.offline import iplot
from sklearn import model_selection
from sklearn.model_selection import train_test_split,
                                    cross_val_score, KFold,
                                    learning_curve, StratifiedKFold,
                                    train_test_split
from sklearn.metrics import confusion_matrix, make_scorer,
                                    accuracy_score
import sklearn.metrics as metrics
from sklearn.ensemble import RandomForestClassifier,
                                    AdaBoostClassifier,
                                    GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC, LinearSVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.cluster import KMeans
from sklearn.model_selection import cross_validate,
                                    RandomizedSearchCV, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer,
                                    HashingVectorizer,
                                    TfidfVectorizer
from nltk.stem import WordNetLemmatizer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import FunctionTransformer, LabelEncoder
from tqdm import tqdm
from google.colab import output
from google.colab import drive
import nltk

tqdm.pandas()
nltk.download("wordnet")
drive.mount('/content/drive')
# %matplotlib inline
```

```python

def compareAccuracy(a, b):
    print('\nCompare Multiple Classifiers: \n')
    print('K-Fold Cross-Validation Accuracy: \n')
    names = []
    models = []
    resultsAccuracy = []
    models.append(('LR', LogisticRegression(max_iter=10000)))
    models.append(('LSVM', LinearSVC(max_iter=10000)))
    models.append(('RF', RandomForestClassifier()))
    for name, model in models:
        model.fit(a, b)
        kfold = model_selection.KFold(n_splits=10, random_state=7, shuffle=True)
        accuracy_results = model_selection.cross_val_score(model, a, b, cv=kfold, scoring='accuracy')
        resultsAccuracy.append(accuracy_results)
        names.append(name)
        accuracyMessage = "%s: %f (%f)" % (name, accuracy_results.mean(), accuracy_results.std())
        print(accuracyMessage)
    # Boxplot
    fig = plt.figure()
    fig.suptitle('Algorithm Comparison: Accuracy')
    ax = fig.add_subplot(111)
    plt.boxplot(resultsAccuracy)
    ax.set_xticklabels(names)
    ax.set_ylabel('Cross-Validation: Accuracy Score')
    plt.show()

lemmatizer = WordNetLemmatizer()
def preprocess(ingredients):
    ingredients_text = ' '.join(ingredients)
    ingredients_text = ingredients_text.lower()
    ingredients_text = ingredients_text.replace('-', '')#wasabe
    ingredients_text = ingredients_text.replace('wasabe', 'wasabi') #for wrong name
    #ingredients_text = ingredients_text.replace('egg whites', 'eggwhites , egg , whites')
    #ingredients_text = ingredients_text.replace('lime juice', 'limejuice')
    #ingredients_text = ingredients_text.replace('clam juice', 'clamjuice')
    #ingredients_text = ingredients_text.replace('lemon juice', 'lemonjuice')
    #ingredients_text = ingredients_text.replace('orange juice', 'orangejuice')
    #ingredients_text = ingredients_text.replace('soy sauce', 'soysauce')
```

```
    ingredients_text = ingredients_text.replace('fish sauce', 'fishsauce')
    #ingredients_text = ingredients_text.replace('sesame oil', 'sesameoil')
    #ingredients_text = ingredients_text.replace('olive oil', 'oliveoil')#vegetable oil corn oil
    #ingredients_text = ingredients_text.replace('vegetable oil', 'vegetableoil')
    #ingredients_text = ingredients_text.replace('corn oil', 'cornoil')#rice wine
    #
    ingredients_text = ingredients_text.replace('coconut cream', 'coconutcream')
    #
    ingredients_text = ingredients_text.replace('yellow onion', 'yellowonion')
    ingredients_text = ingredients_text.replace('cream cheese', 'creamcheese')
    ingredients_text = ingredients_text.replace('baby spinach', 'babyspinach')
    ingredients_text = ingredients_text.replace('coriander seeds', 'corianderseeds')
    ingredients_text = ingredients_text.replace('corn tortillas', 'corntortillas')
    ingredients_text = ingredients_text.replace('rice cakes', 'ricecakes')
    words = []
    for word in ingredients_text.split():
        if re.findall('[0-9]', word): continue
        if len(word) <= 2: continue
        if '    ' in word: continue
        word = lemmatizer.lemmatize(word)
        if len(word) > 0: words.append(word)
    return ' '.join(words)

def fourError(X, Y, model):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0, stratify=Y)

    Train_x, TrainDev_x, Train_y, TrainDev_y = train_test_split(X_train, Y_train, test_size=0.14, random_state=0, stratify=Y_train)
    Dev_x, Test_x, Dev_y, Test_y = train_test_split(X_test, Y_test, test_size=0.5, random_state=0, stratify=Y_test)

    scores = cross_val_score(model, Train_x, Train_y, cv=5, scoring="accuracy")
```

```
    print("Train Error,   e1: ", scores.mean(), "\n")
    model.fit(Train_x, Train_y)

    y_true, trainDev_pred = TrainDev_y, model.predict(TrainDev_x)

    print("Train-Train Dev,   e2:", metrics.mean_squared_error(
                                TrainDev_y, trainDev_pred),"\n")
    print("SVM Accuracy: ", 1 - metrics.mean_squared_error(
                                TrainDev_y, trainDev_pred))
    print( '\nClassification report\n' )
    print(classification_report(y_true, trainDev_pred))

    y_true, dev_pred = Dev_y, model.predict(Dev_x)
    print("Train-Dev,   e3", metrics.mean_squared_error(Dev_y,
                                dev_pred),"\n")
    print("SVM Accuracy: ", 1 - metrics.mean_squared_error(Dev_y,
                                dev_pred))
    print( '\nClassification report\n' )
    print(classification_report(y_true, dev_pred))

    y_true, test_pred = Test_y, model.predict(Test_x)
    print("Train-Test,   e4: ", metrics.mean_squared_error(Test_y,
                                test_pred),"\n")
    print("SVM Accuracy: ", 1 - metrics.mean_squared_error(Test_y,
                                test_pred))
    print( '\nClassification report\n' )
    print(classification_report(y_true, test_pred))

def baseModelComparision(x_train, y_train):
    svc_linear = LinearSVC(max_iter=-1)
    svc = SVC(cache_size=1000, max_iter=-1)
    forest = RandomForestClassifier()
    kmeans = KMeans()

    #accuracy_results1 = cross_val_score(svc_linear, x_train,
                                y_train, scoring='accuracy')
    #print("1 {}".format(accuracy_results1))

    accuracy_results4 = cross_val_score(svc, x_train, y_train,
                                scoring='accuracy')
    print("4 {}".format(accuracy_results4))

    accuracy_results2 = cross_val_score(forest, x_train, y_train,
                                scoring='accuracy')
    accuracy_results3 = cross_val_score(kmeans, x_train, y_train,
                                scoring='accuracy')

    print("2 {}".format(accuracy_results2))
    print("3 {}".format(accuracy_results3))

```

```
def logisticModel(x_train, y_train):

    x_train, x_test, y_train, y_test = train_test_split(x_train,
                                        y_train, test_size=0.3, stratify=
                                        y_train, random_state=0)

    clf = LogisticRegression(n_jobs=-1, multi_class='ovr', solver='
                                        saga', max_iter=2000)
    grid_values = {'penalty': ['l1', 'l2'],
                   'C':[1,5,10],
                   }

    grid = GridSearchCV(clf, param_grid = grid_values, scoring = '
                                        accuracy')
    grid.fit(x_train, y_train)

    print(grid.bestestimator)



    # 78.6070
    model = LogisticRegression(penalty='l2', C=5, n_jobs=-1,
                                        multi_class='ovr', solver='saga',
                                         max_iter=10000)
    model.fit(x_train, y_train)
    accuracy_results = cross_val_score(model, x_train, y_train, cv=
                                        5, scoring='accuracy')
    print("Logistic Regression Cross-Validation Accuracy: ",
                                        accuracy_results.mean())

    clf = AdaBoostClassifier(base_estimator = model, random_state=0
                                        , n_estimators=100, learning_rate
                                        =1)
    clf.fit(x_train, y_train)
    accuracy_results = cross_val_score(model, x_train, y_train, cv=
                                        5, scoring='accuracy')
    print("AdaBoost- Logistic Regression Cross-Validation Accuracy:
                                         ", accuracy_results.mean())

def MultinomialNaive(x_train, y_train):
    x_train, x_test, y_train, y_test = train_test_split(x_train,
                                        y_train, test_size=0.3, stratify=
                                        y_train, random_state=0)

    model = MultinomialNB()
    cv = cross_val_score(model, x_train, y_train, cv=5, scoring='
                                        accuracy')
    print("5-Fold: ", cv.mean()*100)

def parameterTuning(x_train, y_train):
```

```
    param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001
                                 ],'kernel': ['rbf', 'poly', '
                                 linear']}

    C = [0.1,1, 10, 100, 200, 250]
    #gamma = [1,0.1,0.01,0.001]
    #kernel = ['rbf', 'poly', 'linear']

    #grid = GridSearchCV(SVC(), param_grid,refit=True, n_jobs=-1)
    #grid.fit(x_train,y_train)

    #print("SVM")
    #print(grid.best_estimator_)
    #print(grid.best_params_)

    #param_grid = {'C': [0.1,1, 10, 100]}

    #grid = GridSearchCV(LinearSVC(), param_grid,refit=True, n_jobs
                                 =-1)
    #grid.fit(x_train,y_train)


    #print("Linear SVM")
    #print(grid.best_estimator_)
    #print(grid.best_params_)

    n_estimators = [100, 300, 500, 800, 1200]
    min_samples_split = [2, 5, 10, 15, 100]
    max_depth = [5, 8, 15, 25, 30]
    min_samples_leaf = [1, 2, 5, 10]

    #hyperF = dict(n_estimators = n_estimators, max_depth =
                                 max_depth,
    #           min_samples_split = min_samples_split,
    #          min_samples_leaf = min_samples_leaf)

    #grid = GridSearchCV(RandomForestClassifier(), hyperF, n_jobs =
                                  -1, refit=True)
    #grid.fit(x_train, y_train)


    #print("Grid Forest")
    #print(grid.best_estimator_)
    #print(grid.best_params_)

    #rand = RandomizedSearchCV(RandomForestClassifier(), hyperF,
                                 n_jobs=-1)
    #rand.fit(x_train, y_train)

    #print("Random Forest")
```

```
    #print(rand.best_estimator_)
    #print(rand.best_params_)

def best_m(x_train, y_train, x_test, y_test):
    estimator = SVC(C=250, kernel='rbf', degree=3, gamma=1.4, coef0
                              =1,
                    shrinking=True, tol=0.001, probability=False,
                              cache_size=1000,
                    class_weight=None, decision_function_shape=None
                              ,
                    random_state=None)
    classifier = OneVsRestClassifier(estimator)
    scores = cross_val_score(classifier, x_train, y_train, cv=5,
                              scoring="accuracy")

    print(scores.mean())
    classifier.fit(x_train, y_train)

    y_pred = label_encoder.inverse_transform(classifier.predict(
                              x_test))
    y_true = label_encoder.inverse_transform(y_test)

    print(f'accuracy score on test data: {accuracy_score(y_true,
                              y_pred)}')

    return classifier

if __name__ == "__main__":
    train_df = pd.read_json('/content/drive/My Drive/CNG562-Project
                              /train.json')
    test_df = pd.read_json('/content/drive/My Drive/CNG562-Project/
                              test.json')
    train=train_df
    test = test_df

train.head(15)

total = train.isnull().sum().sort_values(ascending = False)
    percent = (train.isnull().sum()/train.isnull().count()*100).
                              sort_values(ascending = False)
    missing_train_data  = pd.concat([total, percent], axis=1, keys=
                              ['Total missing', 'Percent
                              missing'])
    print("              # of Rows, Columns:",train.shape)
    print(missing_train_data.head())

color_theme = dict(color = ['rgba(221,160,221,1)','rgba(169,169,169
                              ,1)','rgba(255,160,122,1)','rgba(
                              176,224,230,1)','rgba(169,169,169
                              ,1)','rgba(255,160,122,1)','rgba(
```

```
                                 176,224,230,1)',
                    'rgba(188,143,143,1)','rgba(221,160,221,1)','rgba(169,169,169,1)','rgba(255,160,122,1)','rgba(176,224,230,1)','rgba(189,183,107,1)','rgba(188,143,143,1)','rgba(221,160,221,1)','rgba(169,169,169,1)','rgba(255,160,122,1)','rgba(176,224,230,1)','rgba(169,169,169,1)','rgba(255,160,122,1)'])
temp = train['cuisine'].value_counts()
trace = go.Bar(y=temp.index[::-1],x=(temp)[::-1],orientation = 'h',marker=color_theme)
layout = go.Layout(title = "Count of recipes per cuisine",xaxis=dict(title='Recipe count',tickfont=dict(size=14,)),
                    yaxis=dict(title='Cuisine',titlefont=dict(size=16),tickfont=dict(size=14)),margin=dict(l=200,))
data = [trace]
fig = go.Figure(data=data, layout=layout)
iplot(fig,filename='basic-bar')

train['x'] = train['ingredients'].progress_apply(lambda ingredients: preprocess(ingredients))
test['x'] = test['ingredients'].progress_apply(lambda ingredients: preprocess(ingredients))
train.head()

tfid_vectorizer = make_pipeline(
    TfidfVectorizer(sublinear_tf=True),
    FunctionTransformer(lambda x: x.astype('float'), validate=False)
)

counter_vectorizer = make_pipeline(
    CountVectorizer(),
    FunctionTransformer(lambda x: x.astype('float'), validate=False)
)

hash_vectorizer = make_pipeline(
    HashingVectorizer(),
    FunctionTransformer(lambda x: x.astype('float'), validate=False)
)

x_train_tfid = tfid_vectorizer.fit_transform(train['x'].values)
x_train_tfid.sort_indices()
x_test_tfid = tfid_vectorizer.transform(test['x'].values)
```

```
x_train_counter = counter_vectorizer.fit_transform(train['x'].values)
x_train_counter.sort_indices()
x_test_counter = counter_vectorizer.transform(test['x'].values)

x_train_hash = hash_vectorizer.fit_transform(train['x'].values)
x_train_hash.sort_indices()
x_test_hash = hash_vectorizer.transform(test['x'].values)

label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(train['cuisine'].values)

tfid_x70_train, tfid_x70_test, tfid_y70_train, tfid_y70_test = train_test_split(x_train_tfid, y_train, test_size=0.3, random_state = 0, stratify=y_train)
counter_x70_train, counter_x70_test, counter_y70_train, counter_y70_test = train_test_split(x_train_counter, y_train, test_size=0.3, random_state = 0, stratify=y_train)
hash_x70_train, hash_x70_test, hash_y70_train, hash_y70_test = train_test_split(x_train_hash, y_train, test_size=0.3, random_state = 0, stratify=y_train)

baseModelComparision(tfid_x70_train, tfid_y70_train)

baseModelComparision(counter_x70_train, counter_y70_train)

baseModelComparision(hash_x70_train, hash_x70_test)

compareAccuracy(x_train_tfid, y_train)

parameterTuning(tfid_x70_train, tfid_y70_train)

model = best_m(tfid_x70_train, tfid_y70_train, tfid_x70_test, tfid_y70_test)
 ada = AdaBoostClassifier(model)
 ada.fit(tfid_x70_train, tfid_x70_train)
 y_pred = label_encoder.inverse_transform(ada.predict(x_train))
 y_true = label_encoder.inverse_transform(y_train)

 print(f'accuracy score on train data: {accuracy_score(y_true, y_pred)}')

```

```
tfid_x70_train , tfid_x70_test , tfid_y70_train , tfid_y70_test =
                                  train_test_split(x_train_tfid ,
                                  y_train , test_size=0.3 ,
                                  random_state = 0, stratify=
                                  y_train)
train_x , train_dev , train_y , test_dev = train_test_split(
                                  tfid_x70_train , tfid_y70_train ,
                                  test_size=0.14 , random_state = 0,
                                   stratify=tfid_y70_train)
test_x , dev_x , test_y , dev_y = train_test_split(tfid_x70_test ,
                                  tfid_y70_test , test_size=0.5 ,
                                  random_state = 0, stratify=
                                  tfid_y70_test)

model = best_m(tfid_x70_train , tfid_y70_train , tfid_x70_test ,
                                  tfid_y70_test)

estimator = SVC(C=250 , kernel='rbf', degree=3 , gamma=1.4 , coef0=1 ,
                    shrinking=True , tol=0.001 , probability=False ,
                                  cache_size=1000 ,
                    class_weight=None , decision_function_shape=None
                                  ,
                    random_state=None)
    classifier = OneVsRestClassifier(estimator)

fourError(x_train_tfid , y_train , classifier)
```