Recipe Nationality Classification

The goal of this project is to predict what country a recipe comes from based on the list of ingredients.

In [118]:
```
import pandas as pd
import json
import nltk
import sklearn
import sklearn.ensemble
import sklearn.discriminant_analysis
import scipy
```

Data Exploration and Feature Extraction

Load data.

In [14]:
```
traindf = pd.read_json(r'C:\Datasets\train.json')
testdf = pd.read_json(r'C:\Datasets\test.json')
```

There are 20 classes in the data set.

In [140]:
```
cuisines = traindf.cuisine.unique().tolist()
cuisines.sort()
cuisines
```

Out[140]:
```
['brazilian',
 'british',
 'cajun_creole',
 'chinese',
 'filipino',
 'french',
 'greek',
 'indian',
 'irish',
 'italian',
 'jamaican',
 'japanese',
 'korean',
 'mexican',
 'moroccan',
 'russian',
 'southern_us',
 'spanish',
 'thai',
 'vietnamese']
```

Rather than treating all ingredients as binary variables, Tf-idf vectorization is used to extract features from the lists of ingredients because it takes into account how common the ingredient is among all recipes and how prominent it is in the individual recipe.

Rather than treating the list as a text, it is treated as a list of tokens. This is achieved by defining the tokenizer and pre-processor as a trivial function that just returns the input.

In [38]:
```
def trivial(input):
    return input

vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(analyzer='word',tokeniz
X = vectorizer.fit_transform(traindf.ingredients)
features = vectorizer.get_feature_names()
```

In [40]:
```
X.shape
```

Out[40]:
```
(39774, 6714)
```

In [61]:
```
features
```

Out[61]:
```
['(    oz.) tomato sauce',
 '(   oz.) tomato paste',
 '(10 oz.) frozen chopped spinach',
 '(10 oz.) frozen chopped spinach, thawed and squeezed dry',
 '(14 oz.) sweetened condensed milk',
 '(14.5 oz.) diced tomatoes',
 '(15 oz.) refried beans',
 '1% low-fat buttermilk',
 '1% low-fat chocolate milk',
 '1% low-fat cottage cheese',
 '1% low-fat milk',
 '2 1/2 to 3 lb. chicken, cut into serving pieces',
 '2% low fat cheddar chees',
 '2% low-fat cottage cheese',
 '2% lowfat greek yogurt',
 '2% milk shredded mozzarella cheese',
 '2% reduced-fat milk',
 '25% less sodium chicken broth',
 '33% less sodium cooked deli ham',
```

There are 6714 unique ingredients in the data set. For now, dimensionality reduction will not be performed, since there are so many classes.

Note that some ingredients include brand or other descriptions. For now these will not be removed, as we do not know if details like the brand name could possibly discriminate between cuisines, especially for common ingredients.

Below is a summary of the top 20 ingredients in each cuisine.

In [59]:

```
for x in traindf.cuisine.unique():
    print(x.upper())
    cuisinefeatures = pd.DataFrame(features, columns=['ingredient'])
    cuisinefeatures['total_tf-idf']=X[traindf.loc[traindf.cuisine==x].index.tolist()
    cuisinefeatures = cuisinefeatures.sort_values('total_tf-idf', ascending=False)
    print(cuisinefeatures.head(20),'\n')
```

```
GREEK
                   ingredient  total_tf-idf
2548  feta cheese crumbles     84.568800
4343  olive oil                81.718261
2547  feta cheese              67.946607
2334  dried oregano            66.337072
5309  salt                     64.838375
2715  fresh lemon juice        57.369160
2126  cucumber                 55.186970
2485  extra-virgin olive oil   51.513861
3690  lemon juice              50.032663
2890  garlic cloves            44.122242
4911  purple onion             41.767618
3135  ground black pepper      40.651064
4569  pepper                   40.549704
2703  fresh dill               38.766062
3683  lemon                    36.131253
2884  garlic                   35.783042
3076  greek yogurt             32.916287
```

Finally, let's also include the normalized number of ingredients as a feature. Some cuisines have many ingredients while others have few.

In [108]:

```
maxingredients = max(traindf.ingredients.apply(len))
numingredients = traindf.ingredients.apply(len).values[:,None]/maxingredients
Xcombined = scipy.sparse.hstack([X,numingredients])
y= traindf.cuisine
```

Modeling

Modeling will be performed using some of the faster multi-classification algorithms from Scikit Learn that support sparse matrices.

Support Vector Machine

In [109]:
```
SVM = sklearn.svm.LinearSVC(class_weight='balanced',max_iter=10000)
Grid = {'C':[0.0001,0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1],'loss':['hinge
SVMmodel = sklearn.model_selection.GridSearchCV(SVM,param_grid=Grid,cv=5)
SVMmodel.fit(Xcombined,y)
for x in list(zip(SVMmodel.cv_results_['mean_test_score'], SVMmodel.cv_results_['par
    print(x)
print(SVMmodel.best_params_)
```

```
(0.6664655302458893, {'C': 0.0001, 'loss': 'hinge'})
(0.46306632473475134, {'C': 0.0001, 'loss': 'squared_hinge'})
(0.7183587268064565, {'C': 0.001, 'loss': 'hinge'})
(0.6796902499120028, {'C': 0.001, 'loss': 'squared_hinge'})
(0.7206215115402022, {'C': 0.01, 'loss': 'hinge'})
(0.7263036154271635, {'C': 0.01, 'loss': 'squared_hinge'})
(0.7402071705134007, {'C': 0.1, 'loss': 'hinge'})
(0.7691708151053452, {'C': 0.1, 'loss': 'squared_hinge'})
(0.752803338864585, {'C': 0.2, 'loss': 'hinge'})
(0.7761603057273596, {'C': 0.2, 'loss': 'squared_hinge'})
(0.7598682556443908, {'C': 0.3, 'loss': 'hinge'})
(0.7762608739377482, {'C': 0.3, 'loss': 'squared_hinge'})
(0.7635641373761753, {'C': 0.4, 'loss': 'hinge'})
(0.7765625785689144, {'C': 0.4, 'loss': 'squared_hinge'})
(0.7662794790566702, {'C': 0.5, 'loss': 'hinge'})
(0.7763865842007341, {'C': 0.5, 'loss': 'squared_hinge'})
(0.7675365816865289, {'C': 0.6, 'loss': 'hinge'})
(0.775783174938402, {'C': 0.6, 'loss': 'squared_hinge'})
(0.7679388545280836, {'C': 0.7, 'loss': 'hinge'})
(0.7751294815708755, {'C': 0.7, 'loss': 'squared_hinge'})
(0.7679388545280836, {'C': 0.8, 'loss': 'hinge'})
(0.7741489415195857, {'C': 0.8, 'loss': 'squared_hinge'})
(0.7687434002111933, {'C': 0.9, 'loss': 'hinge'})
(0.7734952481520592, {'C': 0.9, 'loss': 'squared_hinge'})
(0.7686176899482073, {'C': 1, 'loss': 'hinge'})
(0.7728666968371298, {'C': 1, 'loss': 'squared_hinge'})
{'C': 0.4, 'loss': 'squared_hinge'}
```

k Nearest Neighbors

In [112]:
```
kNN = sklearn.neighbors.KNeighborsClassifier()
Grid = {'n_neighbors':[5, 10, 50, 100, 150]}
kNNmodel = sklearn.model_selection.GridSearchCV(kNN,param_grid=Grid,cv=5)
kNNmodel.fit(Xcombined,y)
for x in list(zip(kNNmodel.cv_results_['mean_test_score'], kNNmodel.cv_results_['par
    print(x)
print(kNNmodel.best_params_)
```

```
(0.7192889827525519, {'n_neighbors': 5})
(0.7392769145673053, {'n_neighbors': 10})
(0.7289183888972696, {'n_neighbors': 50})
(0.7123749182883291, {'n_neighbors': 100})
(0.6979433800975512, {'n_neighbors': 150})
{'n_neighbors': 10}
```

In [121]:
```
kNN = sklearn.neighbors.KNeighborsClassifier()
Grid = {'n_neighbors':[5, 8, 10, 15, 20, 30]}
kNNmodel = sklearn.model_selection.GridSearchCV(kNN,param_grid=Grid,cv=5)
kNNmodel.fit(Xcombined,y)
for x in list(zip(kNNmodel.cv_results_['mean_test_score'], kNNmodel.cv_results_['par
    print(x)
print(kNNmodel.best_params_)
```

```
(0.7192889827525519, {'n_neighbors': 5})
(0.7346004927842309, {'n_neighbors': 8})
(0.7392769145673053, {'n_neighbors': 10})
(0.7420676824055916, {'n_neighbors': 15})
(0.7415145572484537, {'n_neighbors': 20})
(0.737139840096545 4, {'n_neighbors': 30})
{'n_neighbors': 15}
```

Random Forest

In [115]:
```
RF = sklearn.ensemble.RandomForestClassifier()
Grid = {'n_estimators':[10, 50, 100, 200, 300],'min_samples_split':[2, 5, 10, 20]}
RFmodel = sklearn.model_selection.GridSearchCV(RF,param_grid=Grid,cv=5)
RFmodel.fit(Xcombined,y)
for x in list(zip(RFmodel.cv_results_['mean_test_score'], RFmodel.cv_results_['param
    print(x)
print(RFmodel.best_params_)
```

```
(0.6573641072057123, {'min_samples_split': 2, 'n_estimators': 10})
(0.7030220747221804, {'min_samples_split': 2, 'n_estimators': 50})
(0.709332729924071, {'min_samples_split': 2, 'n_estimators': 100})
(0.7111429577110675, {'min_samples_split': 2, 'n_estimators': 200})
(0.7106652587117213, {'min_samples_split': 2, 'n_estimators': 300})
(0.6754412430230804, {'min_samples_split': 5, 'n_estimators': 10})
(0.7039020465630814, {'min_samples_split': 5, 'n_estimators': 50})
(0.7067933826117564, {'min_samples_split': 5, 'n_estimators': 100})
(0.7092070196610851, {'min_samples_split': 5, 'n_estimators': 200})
(0.7097350027656257, {'min_samples_split': 5, 'n_estimators': 300})
(0.6731533162367376, {'min_samples_split': 10, 'n_estimators': 10})
(0.7009352843566149, {'min_samples_split': 10, 'n_estimators': 50})
(0.7019158244079047, {'min_samples_split': 10, 'n_estimators': 100})
(0.7049580127721627, {'min_samples_split': 10, 'n_estimators': 200})
(0.7047568763513853, {'min_samples_split': 10, 'n_estimators': 300})
(0.6715442248705185, {'min_samples_split': 20, 'n_estimators': 10})
(0.6967868456780811, {'min_samples_split': 20, 'n_estimators': 50})
(0.6971136923618444, {'min_samples_split': 20, 'n_estimators': 100})
(0.69937647709559, {'min_samples_split': 20, 'n_estimators': 200})
(0.7005330115150601, {'min_samples_split': 20, 'n_estimators': 300})
{'min_samples_split': 2, 'n_estimators': 200}
```

Final Model

Of the models tested, SVM has the best accuracy for cross validation. Therefore it will be used as the final model.

In [122]:

```
SVM = sklearn.svm.LinearSVC(class_weight='balanced',max_iter=10000, C=0.4)
SVM.fit(Xcombined,y)
```

...

The 20 ingredients with the highest coefficients are listed below for each cuisine.

In [156]:

```
for i in range(20):
    print(cuisines[i])
    coefdf = pd.DataFrame(features,columns=['ingredient'])
    coefdf['coefficient']=pd.DataFrame(SVM.coef_[i].tolist()).loc[:,0]
    coefdf = coefdf.sort_values('coefficient', ascending=False)
    print(coefdf.iloc[:20,:],'\n')
```

```
brazilian
                   ingredient  coefficient
1260  cachaca                     4.346554
3259  hearts of palm              2.931589
3984  manioc flour                2.848472
6036  tapioca flour               2.541129
708   aÃ§ai                       2.448965
2213  dende oil                   2.446963
2291  dried black beans           2.405388
4428  palm oil                    2.391212
1691  chocolate sprinkles         2.320390
5978  sweetened condensed milk    2.247770
6038  tapioca starch              2.196635
5815  starch                      2.081849
5851  stone-ground cornmeal       2.037575
939   black beans                 2.020944
2357  dried shrimp                1.977200
1830  coconut milk                1.956501
6290  unsweetened coconut milk    1.949795
```

The confusion matrix is below.

In [141]:
```
pd.DataFrame(sklearn.metrics.confusion_matrix(y,SVM.predict(Xcombined)),index=cuisin
```

Out[141]:

	brazilian	british	cajun_creole	chinese	filipino	french	greek	indian	irish	italian	jama
brazilian	436	1	2	0	3	3	0	3	1	0	
british	1	674	2	1	1	20	0	9	39	3	
cajun_creole	5	6	1376	1	0	20	0	2	3	19	
chinese	4	4	6	2401	28	13	1	5	1	8	
filipino	6	2	1	20	677	1	0	5	2	4	
french	6	62	19	3	6	2022	35	10	49	201	
greek	3	3	0	2	1	11	1063	6	0	41	
indian	10	8	3	6	5	3	20	2800	10	2	
irish	0	28	0	1	1	14	3	2	584	4	
italian	14	69	43	7	14	312	155	14	53	6749	
jamaican	1	2	0	0	1	1	1	3	1	1	
japanese	4	3	2	69	10	8	1	95	1	3	
korean	0	1	0	19	3	1	0	0	1	1	
mexican	43	15	26	11	27	53	13	13	5	47	
moroccan	1	1	1	0	0	2	7	15	0	4	
russian	1	7	1	0	1	10	3	2	4	1	
southern_us	17	82	199	11	30	96	16	19	60	63	
spanish	6	6	10	2	5	49	13	2	5	36	
thai	5	0	0	43	11	0	2	27	0	3	
vietnamese	5	1	0	41	11	3	1	2	0	2	

Understandably recipes from similar cuisines are most often confused (cajun and southern, chinese and japanese, etc.).

Prediction for Test Data Set

In [157]:
```
Xtest = vectorizer.transform(testdf.ingredients)
testnumingredients = testdf.ingredients.apply(len).values[:,None]/maxingredients
Xtestcombined = scipy.sparse.hstack([Xtest,testnumingredients])
ytest = SVM.predict(Xtestcombined)
```

In [162]:
```
pd.DataFrame(ytest,index=testdf.id, columns=['cuisine']).to_csv('C:\Datasets\Cuisine
```