task02-cuisine-map

September 23, 2018

loganjtravis@gmail.com (Logan Travis)

In [1]:
```
%%capture --no-stdout

# Imports; captures errors to supress warnings about changing
# import syntax
import random
import gensim.models as models, \
       gensim.matutils as matutils, \
       gensim.corpora as corpora
import matplotlib.pyplot as plot
import nltk
import numpy as np
import pandas as pd
import scipy as sp
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import CountVectorizer, \
                                            TfidfTransformer
from sklearn.metrics.pairwise import cosine_similarity, \
                                     euclidean_distances, \
                                     paired_cosine_distances
from sklearn.preprocessing import normalize
```

In [2]:
```
# Set random seed for repeatability
random.seed(42)
```

In [3]:
```
# Set matplotlib to inline to preserve images in PDF
%matplotlib inline
```

1 Summary

From course page Week 2 > Task 2 Information > Task 2 Overview:

> The goal of this task is to mine the data set to construct a cuisine map to visually understand the landscape of different types of cuisines and their similarities. The cuisine map can help users understand what cuisines are available and their relations, which allows for the discovery of new cuisines, thus facilitating exploration of unfamiliar

cuisines. You can see a sample set of reviews from all the restaurants for a cuisine, but you are strongly encouraged to experiment with your own set of cuisines if you have time.

Instructions Some questions to consider when building the cuisine map are the following:

1. What's the best way of representing a cuisine? If all we know about a cuisine is just the name, then there is nothing we can do. However, if we can associate a cuisine with the restaurants offering the cuisine and all the information about the restaurants, particularly reviews, then we will have a basis to characterize cuisines and assess their similarity. Since review text contains a lot of useful information about a cuisine, a natural question is: what's the best way to represent a cuisine with review text data? Are some words more important in representing a cuisine than others?
2. What's the best way of computing the similarity of two cuisines? Assuming that two cuisines can each be represented by their corresponding reviews, how should we compute their similarity?
3. What's the best way of clustering cuisines? Clustering of cuisines can help reveal major categories of cuisines. How would the number of clusters impact the utility of your results for understanding cuisine categories? How does a clustering algorithm affect the visualization of the cuisine map?
4. Is your cuisine map actually useful to at least some people? In what way? If it's not useful, how might you be able to improve it to make it more useful?

2 Get Yelp Review Data Set

I cleaned the Yelp review data and extraced cuisines from the business data set in separate notebooks. Loading saved data to shorten this report.

In [4]:
```
# Set paths to data source, work in process ("WIP"), and output
PATH_SOURCE = "source/"
PATH_WIP = "wip/"
PATH_OUTPUT = "output/"

# Set file paths
PATH_SOURCE_YELP_REVIEWS = PATH_SOURCE + \
        "yelp_academic_dataset_review.pkl.gzip"
PATH_SOURCE_YELP_CUISINES = PATH_SOURCE + "yelp_academic_dataset_cuisine.csv"
PATH_SOURCE_YELP_REST_TO_CUISINES = PATH_SOURCE + \
        "yelp_academic_dataset_restaurant_to_cuisine.pkl.gzip"
```

In [5]:
```
# Read saved data
dfYelpReviews = pd.read_pickle(PATH_SOURCE_YELP_REVIEWS)
dfYelpRestToCuis = pd.read_pickle(PATH_SOURCE_YELP_REST_TO_CUISINES)
cuisines = pd.read_csv(PATH_SOURCE_YELP_CUISINES, names=["cuisine"])
```

In [6]:
```
# Join (inner) reviews to restaurants
dfYelpReviews = dfYelpReviews.join(dfYelpRestToCuis, \
```

```
                                    on="business_id", \
                                    how="inner", \
                                    rsuffix="_business")
```

3 Determine Cuisine Similarity via Term Frequencies

I calculate the term frequencies across cuisines both with and without inverse-document frequency as an initial comparison. These methods take comparatively less time and also precompute data necessary for more advanced similarity comparisons like my seeded-LDA topic approach.

3.1 TF Parameters

I found the settings below worked well when extracting topics for the week one assignment:

- Limit maximum terms to 10,000. This is an extreme upper limit. The SciKit Learn `TfidfVectorizer` class (link to documentation) never yielded more than 5,000 terms based on my other parameters.
- Exclude terms appearing in more than 50% of documents. These add little value for differntiating topics.
- Exclude terms appearing in less than 0.1% of documents. I tested many settings for this parameter ranging down to 2 documents and up to 10% of all documents. The Yelp reviews include numerious limited-use terms (e.g., people and place names) and I found it difficult to interpret the topics with too many present.

```
In [7]: # Set token limit
        MAX_FEATURES = 10000

        # Set document frequency ceiling; topic analysis will ignore
        # words found in more documents
        MAX_DF = 0.5

        # Set document frequency floor; topic analysis will ignore
        # words found in fewer document
        MIN_DF = 0.001
```

```
In [8]: class MyTokenizer:
            def __init__(self):
                """String tokenizer utilizing lemmatizing and stemming."""
                self.wnl = nltk.stem.WordNetLemmatizer()

            def __call__(self, document):
                """Return tokens from a string."""
                return [self.wnl.lemmatize(token) for \
                                token in nltk.word_tokenize(document)]
```

3.2 Excessive Data Set Size

I worked on a 50% sample of the Yelp reviews for restaurants. The full data set proved too large for my machine's memory.

```
In [9]: # Set working dataframe to a 50% sample of the full data set;
        # too large otherwise
        df = dfYelpReviews.sample(frac=0.5)

In [10]: # Reset index to simple integer; simplifies document selection
         # in TF matrix
         df.reset_index(inplace=True)

In [11]: # Create TF vectorizer
         tf = CountVectorizer(max_features=MAX_FEATURES, max_df=MAX_DF, \
                              min_df=MIN_DF, stop_words="english", \
                              tokenizer=MyTokenizer())

In [12]: %%time

         # Calculate term frequencies
         docTerms = tf.fit_transform(df.text)

CPU times: user 8min 6s, sys: 5.14 s, total: 8min 11s
Wall time: 8min 16s


In [13]: # Print token matrix shape
         print("Found {0[1]:,} tokens in {0[0]:,} documents".format(docTerms.shape))

Found 4,259 tokens in 353,323 documents
```

3.3 Cuisine TF Cosine Similarity

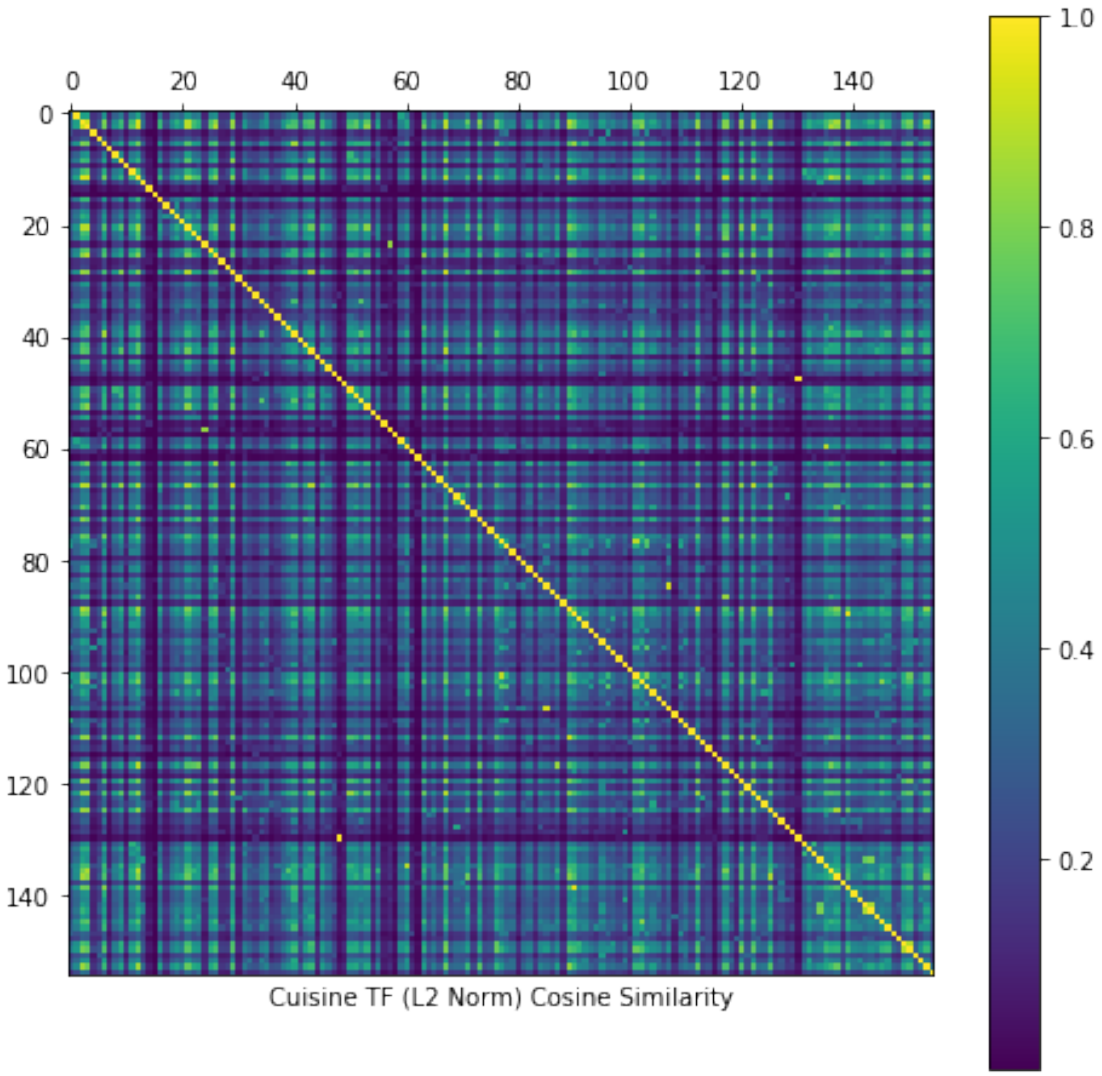
```
In [14]: # Sort cuisines alphabetically
         cuisines = cuisines.sort_values(by="cuisine").reset_index(drop=True)

In [15]: %%time

         # Sum term frequencies across documents for each cuisine
         cuisineTerms = sp.sparse.coo_matrix((0, docTerms.shape[1]))
         for c in cuisines.cuisine:
             idxs = df[df.categories.apply(lambda cats: c in cats)].index
             cuisineTerms = sp.sparse.vstack([cuisineTerms, docTerms[idxs, ].sum(axis=0)])

CPU times: user 8min 32s, sys: 46.4 s, total: 9min 18s
Wall time: 8min 58s
```

In [16]:
```
# Convert to CSR sparse matrix to facility row-wise operations
cuisineTerms = cuisineTerms.tocsr()
```

In [17]:
```
# Normalize term frequencies by cuisine for initial graph
cuisineTermsL2Norm = normalize(cuisineTerms, axis=0)
```

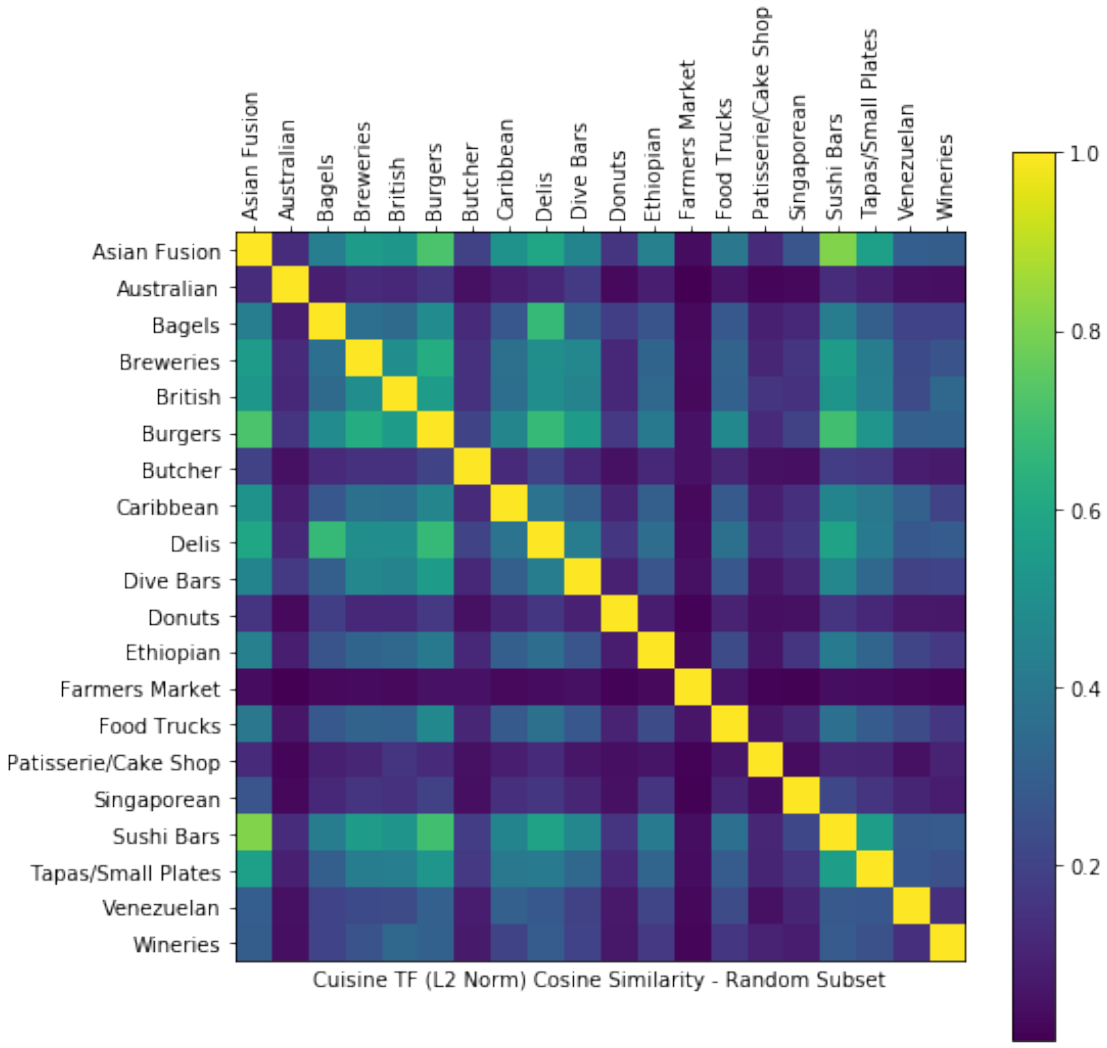
In [18]:
```
# Define graphing function for reusability
def graphSimilarity(matrix, labels, title, display_labels=True, colors=None, \
                    matrix_under=None, alpha_over=0.2):
    """Graph a similarity matrix."""
    fig, ax = plot.subplots()
    fig.set_size_inches(8, 8)

    # Generate matrix image with or without underlay
    cmap = colors if not None else plot.cm.get_cmap("viridis")
    if matrix_under is None:
        cax = ax.matshow(matrix, cmap=cmap)
    else:
        cmap_over = plot.get_cmap("binary_r")
        cax_under = ax.matshow(matrix_under, cmap=cmap, vmin=0, vmax=1)
        cax = ax.matshow(matrix, cmap=cmap_over, alpha=alpha_over)


    # Configure axes
    ax.xaxis.tick_top()
    if display_labels:
        plot.xticks(range(matrix.shape[0]), labels, rotation=90)
        plot.yticks(range(matrix.shape[0]), labels)

    # Add title (as x-axis label) and color bar
    ax.set_xlabel(title)
    fig.colorbar(cax)

    # Return plot
    return fig, ax
```

In [19]:
```
# Plot cosine similarity for all cuisine TFs
matrix = cosine_similarity(cuisineTermsL2Norm)
labels = cuisines.cuisine
_, _ = graphSimilarity(matrix, labels, \
                       "Cuisine TF (L2 Norm) Cosine Similarity", \
                       display_labels=False)
plot.show()
```



While quick, calculating cosine similarity from intra-topic normalized term frequencies yields poor results. Many cuisines appear unrelated to all others as indicated by the dark bands (i.e., cosine similarity near zero).

```
In [20]: # Sample cuisines for detailed view
         SAMPLE_20 = sorted(random.sample(range(cuisines.shape[0]), 20))

In [21]: # Plot cosine similarity for random 20 cuisine TFs
         matrix = cosine_similarity(cuisineTermsL2Norm[SAMPLE_20, :])
         labels = cuisines.iloc[SAMPLE_20, ].cuisine
         _, _ = graphSimilarity(matrix, labels, \
                                "Cuisine TF (L2 Norm) Cosine Similarity - Random Subset", \
                                display_labels=True)
         plot.show()
```



Zooming in on a sample of cuisines shows some expected similarities: Delis to Bagels and Asian Fusion to Sushi Bars. Still, most cuisines appear unrelated to each other.

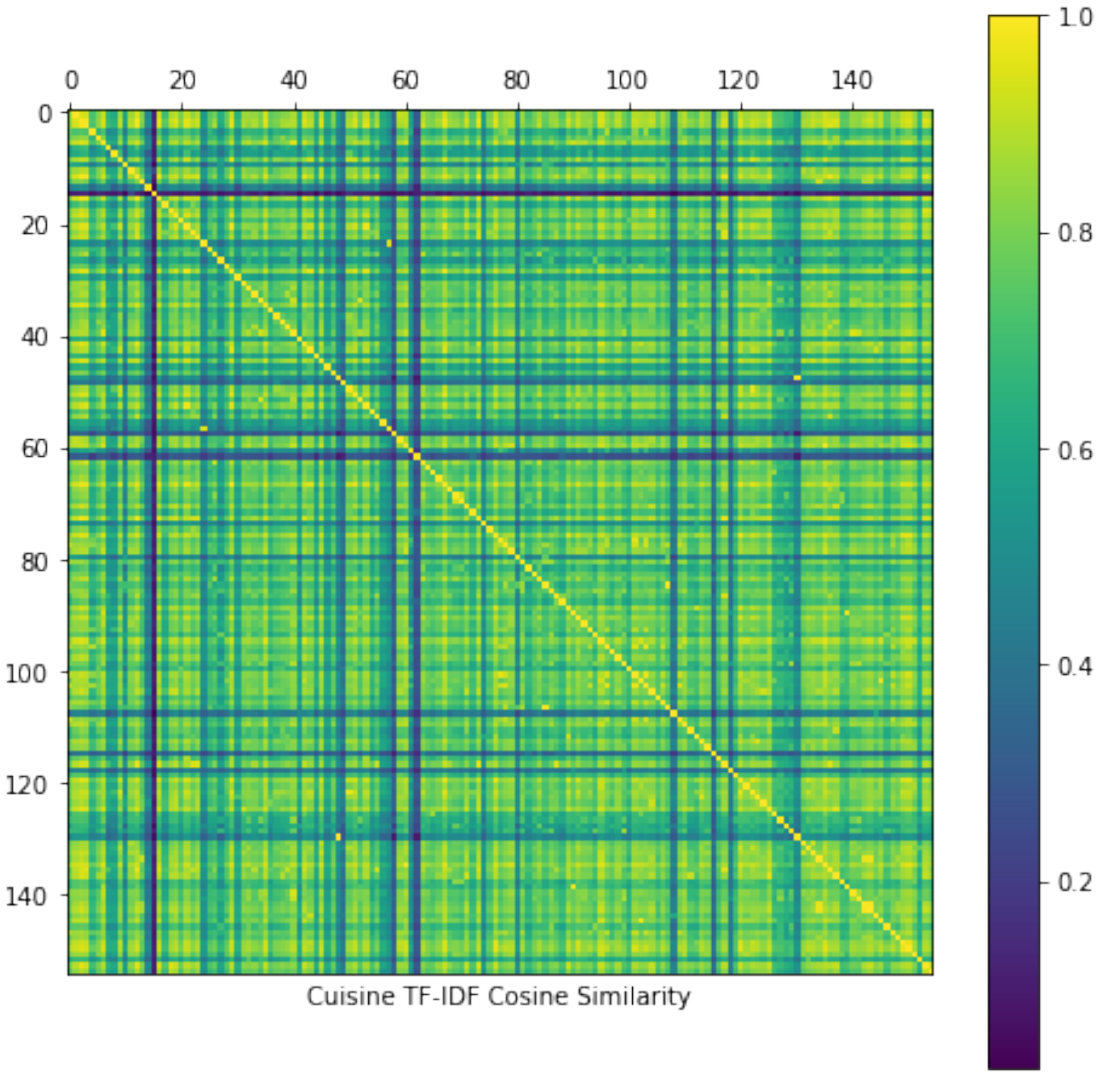
3.4 Cuisine TF-IDF Cosine Similarity

Applying IDF weighting should improve the cuisine comparison. Note that "document" in IDF does not mean the individual reviews. It instead means the topics. I essentially create an imaginary document summing all the terms seen in reviews for a given topic.

I considered calculating TF-IDF against the reviews. That would yield better per-term weighting. However, I do not know how to consolidate the review-driven TF-IDF results across topics.

```
In [22]: # Create TF-IDF transformer from counts
         tfToTfidf = TfidfTransformer(norm="l2", use_idf=True)

In [23]: # Calculate TF-IDF for topics
         cuisineTermsIDF = tfToTfidf.fit_transform(cuisineTerms)
```

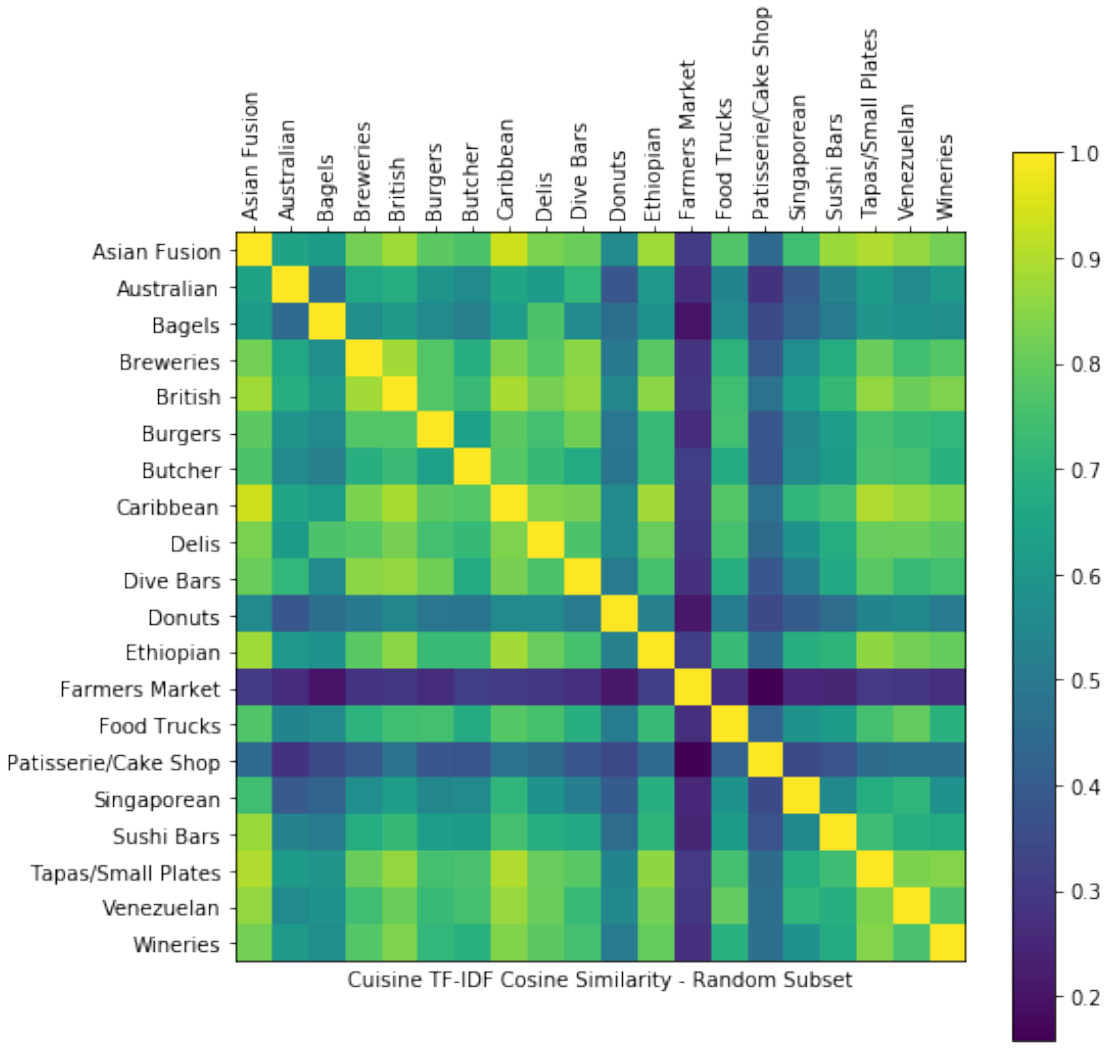
```
In [24]: # Plot cosine similarity for all cuisines TF-IDF
         matrix = cosine_similarity(cuisineTermsIDF)
         labels = cuisines.cuisine
         _, _ = graphSimilarity(matrix, labels, \
                                "Cuisine TF-IDF Cosine Similarity",
                                display_labels=False)
         plot.show()
```



Immediately we see the IDF produces a better similarity spread. Many cuisines relate closely to one another - as expected in the real world - while some retain their distinctness.

```
In [25]: # Plot cosine similarity for random 20 cuisine TF-IDFs
         matrix = cosine_similarity(cuisineTermsIDF[SAMPLE_20, :])
         labels = cuisines.iloc[SAMPLE_20, ].cuisine
         _, _ = graphSimilarity(matrix, labels, \
```

```
                                "Cuisine TF-IDF Cosine Similarity - Random Subset", \
                                display_labels=True)
plot.show()
```



The same sampled cuisines reveal new similarities after applying IDF: Asian Fusion to Caribbean and Ethiopian to Tapas/Small Plates. The IDF also highlights the not-quite-cuisines like Bagels, Donuts, Famers Market, and Patisserie/Cake Shop.

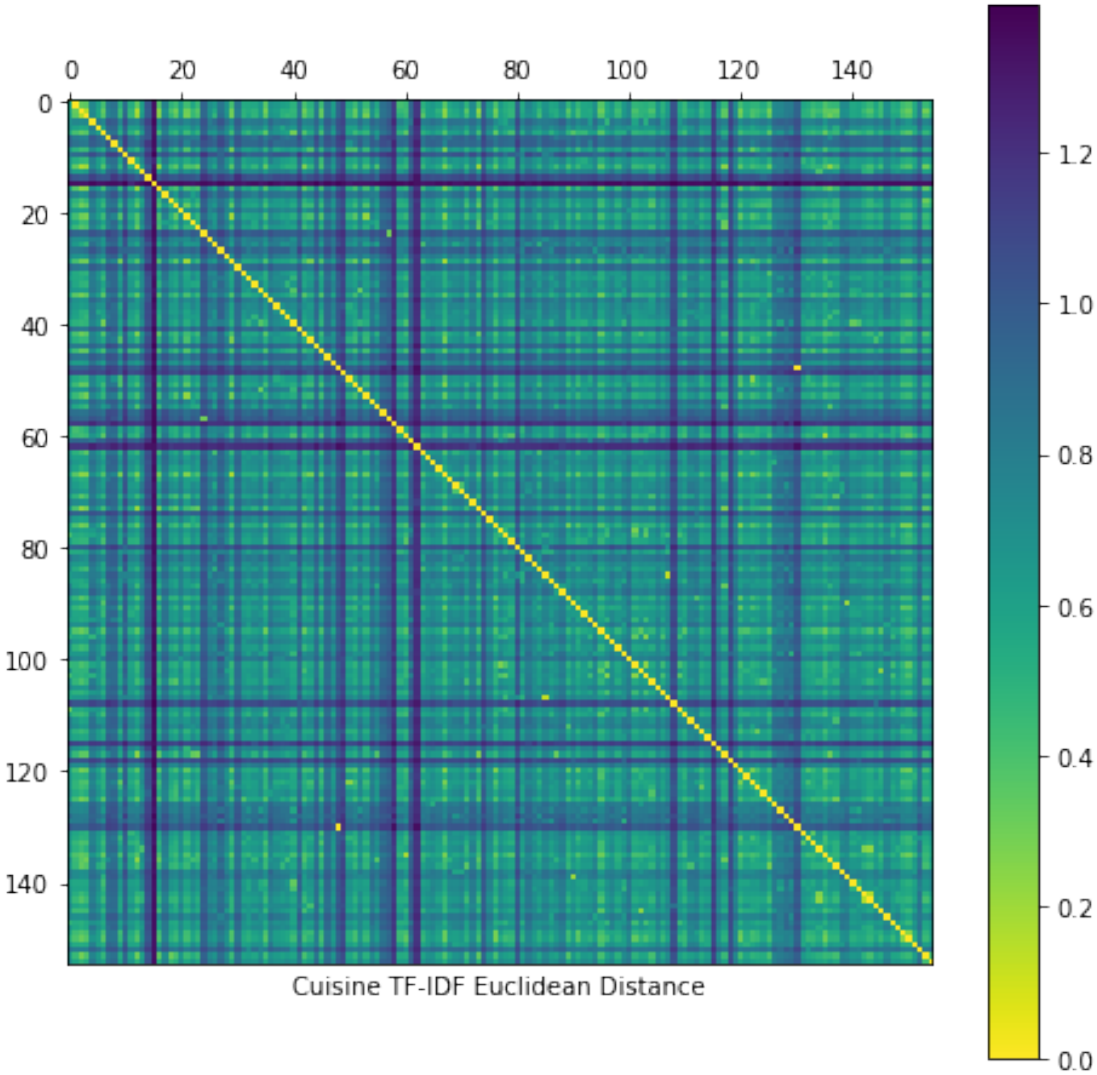
Not all results makes intuitive sense. Delis, a not-quite-cuisine, displays moderate similarity with most cuisines. So too does Wineries. Their reviews may exhibit enough common terms to explain the similarities. As an example, many restaurants sell alcohol so might relate to Wineries regardless of cuisine.

I still think I can improve the cuisine map.

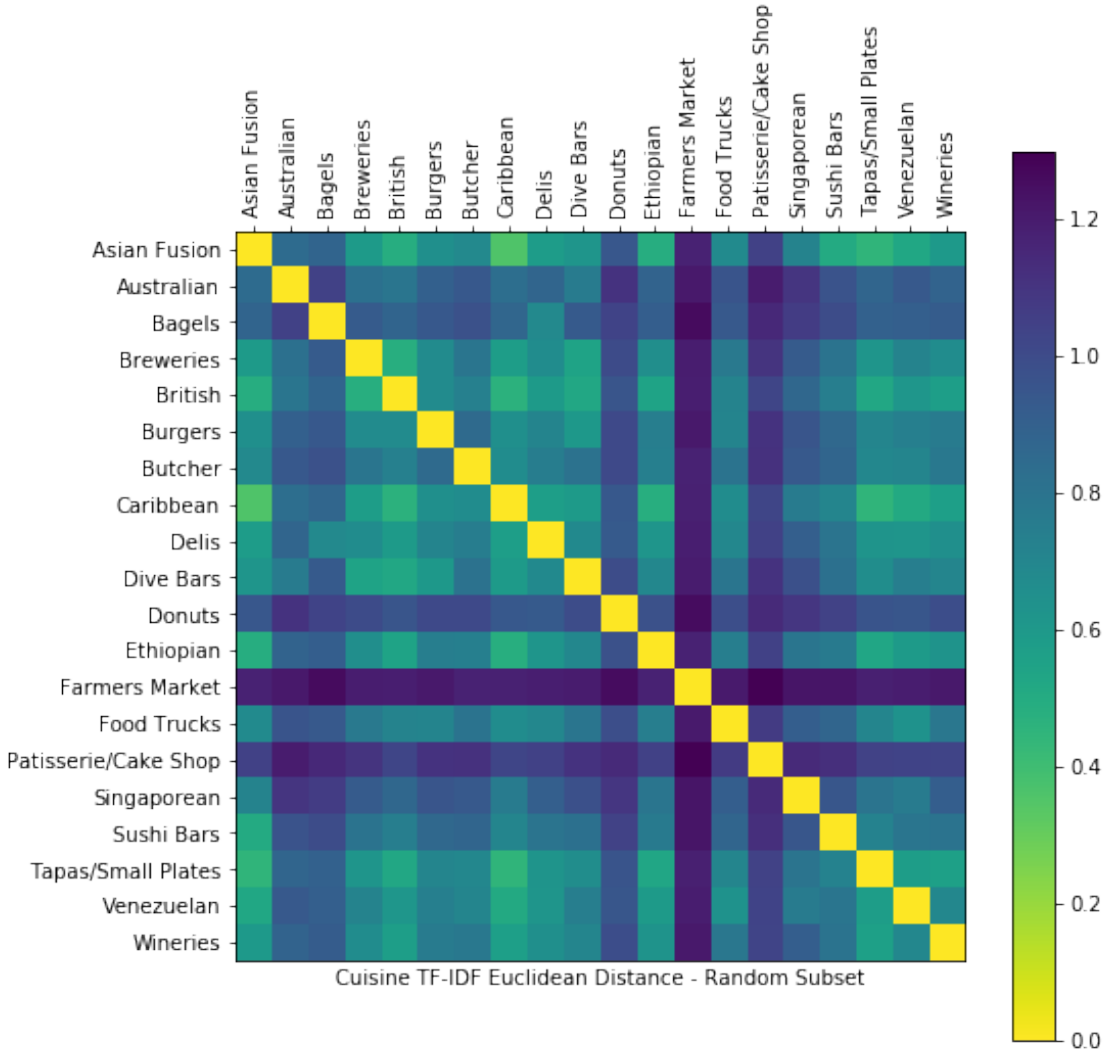
3.5 Cuisine TF-IDF Euclidian Distance

I perform a quick sanity check by comparing TF-IDF accross cuisines using Euclidean distance. It produces similar results to cosine similarity (with color scale reversed). That makes sense since all the TF-IDF vectors point into the same sector (i.e., all positive values) and have similar magnitudes due to normalization.

```
In [26]: # Get revers viridis color map
         cmap_viridis_r = plot.get_cmap("viridis_r")

In [27]: # Plot euclidian distance (reverse color map) for all
         # cuisine TF-IDF
         matrix = euclidean_distances(cuisineTermsIDF)
         labels = cuisines.cuisine
         _, _ = graphSimilarity(matrix, labels, \
                                "Cuisine TF-IDF Euclidean Distance", \
                                display_labels=False, colors=cmap_viridis_r)
         plot.show()
```



In [28]:
```
# Plot euclidian distance (reverse color map) for random
# 20 cuisine TF-IDFs
matrix = euclidean_distances(cuisineTermsIDF[SAMPLE_20, :])
labels = cuisines.iloc[SAMPLE_20, ].cuisine
_, _ = graphSimilarity(matrix, labels, \
                       "Cuisine TF-IDF Euclidean Distance - Random Subset", \
                       display_labels=True, colors=cmap_viridis_r)
plot.show()
```



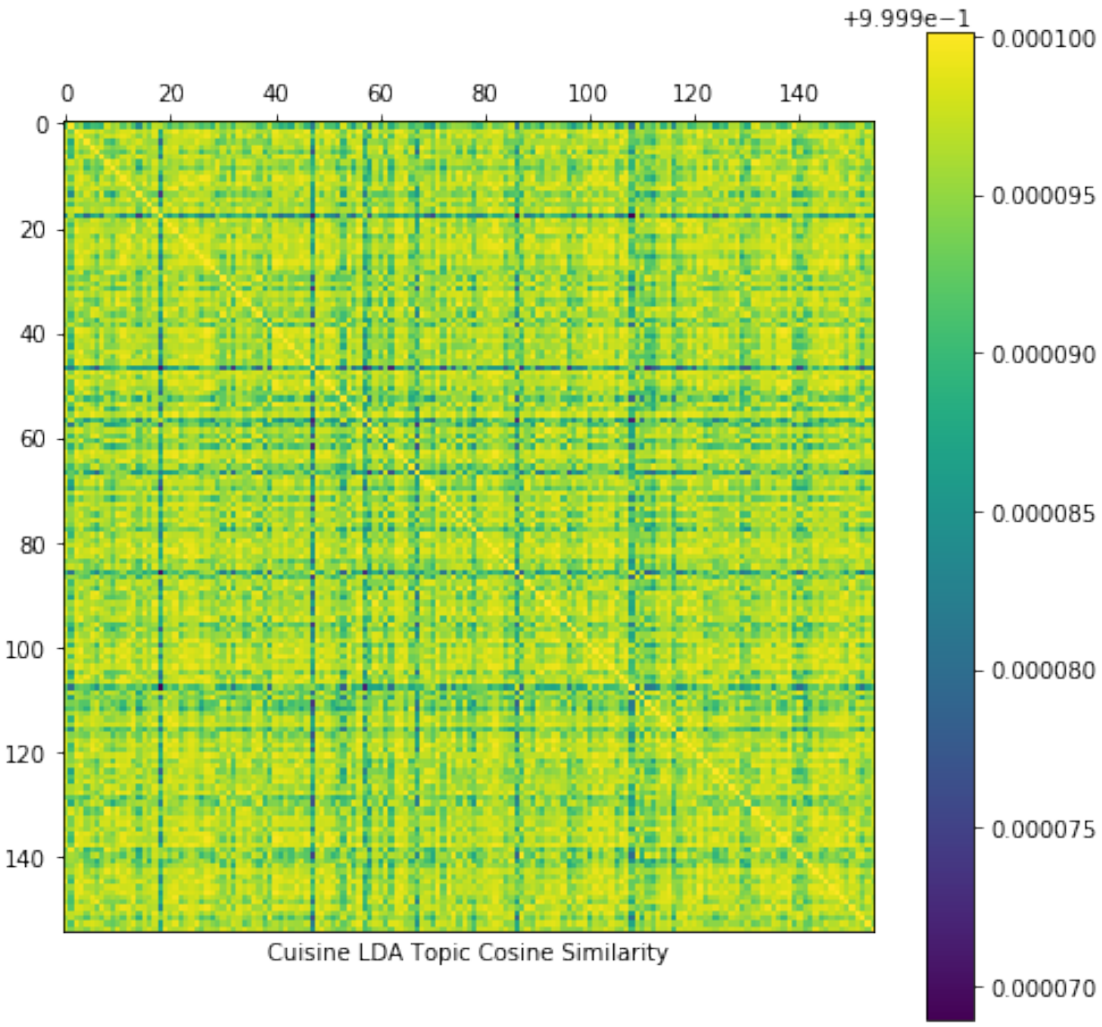
4 Determine Cuisine Similarity via LDA

Improving the cuisine map requires improving the text-based representation of each cuisine. LDA offers one potential method: Attempt to discover topics that match cuisines. I say "attempt" because LDA is an unsuperived algorithm; I cannot provide it the cuisines as ground truths.

I research extensions of LDA with supervised or "guided" learning. See Google Scholar Articles Related to Labeled LDA. Unfortunately, I found myself time constrained so choose instead to seed the regular LDA with cuisine TFs as a-priori probabilities.

```
In [29]: # Set number of topics
         NUM_TOPICS = cuisines.shape[0]

In [30]: # Calculate TF-IDF for terms
         docTermsIDF = tfToTfidf.fit_transform(docTerms)

In [31]: # Convert GenSim corpus from token matrix
         corpus = matutils.Sparse2Corpus(docTermsIDF, documents_columns=False)

In [32]: # Create a GenSim dictionary for documents; Note: Passes the
         # vectorizer tokens as a single "document".
         dictionary = corpora.Dictionary([tf.get_feature_names()])

In [33]: %%time
         %%capture --no-stdout

         # Find topic using LDA with a-priori from cuisine TF
         lda = models.ldamulticore.LdaMulticore(corpus, \
                                                num_topics=NUM_TOPICS,
                                                id2word=dict(dictionary.items()), \
                                                eta=cuisineTerms.todense().tolist()) # Memory in

CPU times: user 3min 34s, sys: 42.8 s, total: 4min 16s
Wall time: 4min 24s


In [34]: # Get topic distribution for each cuisine
         rows, cols, vals = [], [], []
         cuisineTerms = cuisineTerms.tocsr()
         for i in range(cuisines.shape[0]):
             row = cuisineTerms.getrow(0)
             bow = list(zip(row.indices, row.data))
             for c, v in lda.get_document_topics(bow, minimum_probability=0):
                 rows.append(i)
                 cols.append(c)
                 vals.append(v)
         cuisineTopics = sp.sparse.csr_matrix((vals, (rows, cols)))

In [35]: # Plot cosine similarity for all cuisine LDA topics
         matrix = cosine_similarity(cuisineTopics)
```

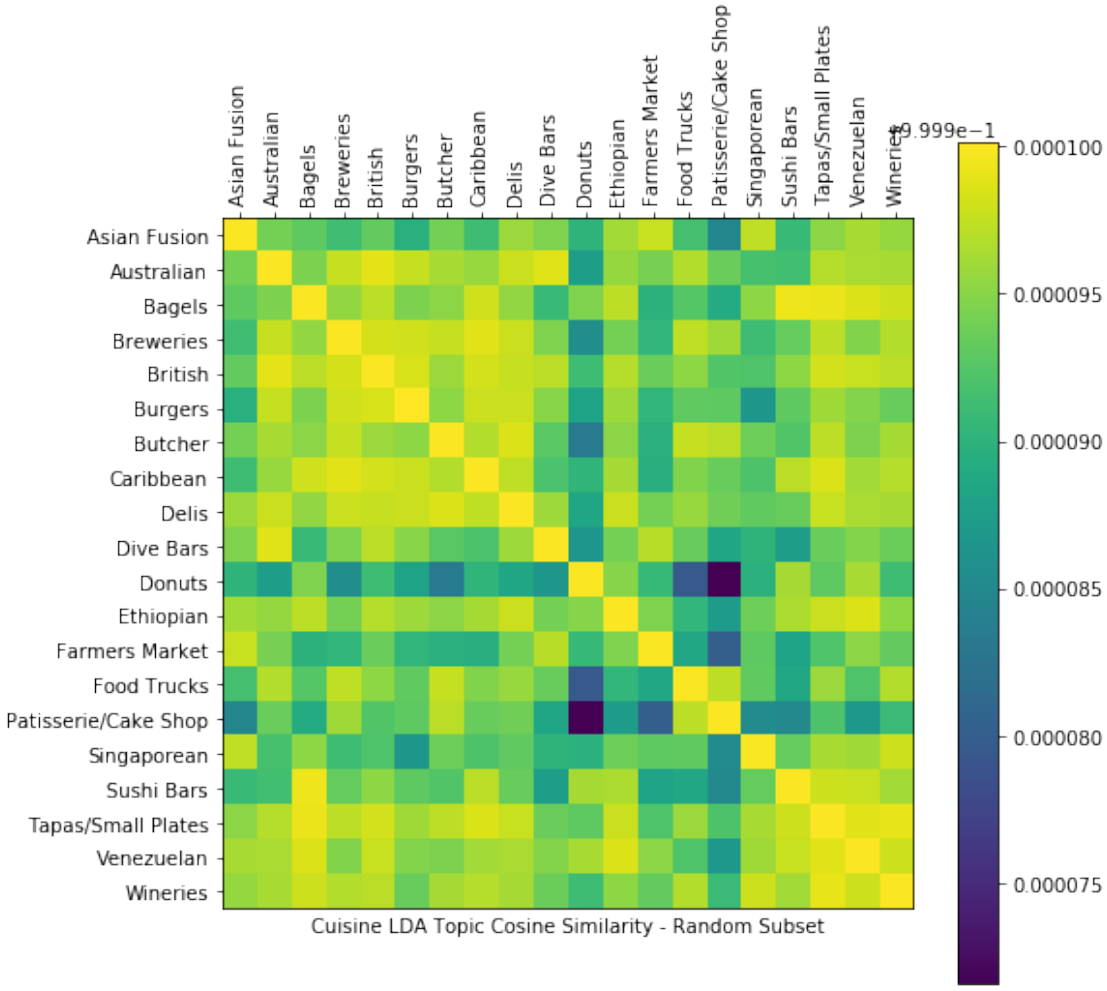
```
labels = cuisines.cuisine
_, _ = graphSimilarity(matrix, labels, \
                       "Cuisine LDA Topic Cosine Similarity", \
                       display_labels=False)
plot.show()
```



Note the scale of the colorbar. It reaches a maximum of just 0.0001 suggesting the seeded-LDA produced topics nearly matching cuisines (i.e. nearly orthogonal). I think the overall effect improves on TF-IDF similarity but only slightly. Notabely, the seeded-LDA approach removes many of the dark bands finding similarities for cuisines that previously appeared completely dissimilar from all others.

```
In [36]: # Plot cosine similarity for random 20 cuisine
         # LDA topics
         matrix = cosine_similarity(cuisineTopics[SAMPLE_20, :])
         labels = cuisines.iloc[SAMPLE_20, ].cuisine
```

```
_, _ = graphSimilarity(matrix, labels, \
                       "Cuisine LDA Topic Cosine Similarity - Random Subset", \
                       display_labels=True)
plot.show()
```



Closer inspection of the sample cuisines shows a "flattened" similarity matrix. Strong dis/similarities appear like Asian Fusion to Tapas/Small Plates (strongly similar) and Delis to Singaporean (strongly dissimilar). Still, the flattened matrix would benefit from better defining such dis/similarities.

5 Cluster Similar Cuisines

I improve the seeded-LDA comparison of cuisines by clustering them and ordering them around those clusters. This took a lot of experimentation. I tried as few as two clusters and as many as 16. I tried varies overlays on the similarity matrix to highlight clusters. I also tried both cosine similarity and euclidean distance again when ordering cuisines within each cluster.

Four clusters and cosine similarity yieleded the best cuisine map.

5.1 Cluster Cuisine Topics with K-Means

In [37]:
```
# Create and train K-Means clustering instance
cuisineTopicClustersKM = KMeans(n_clusters=4, random_state=42).fit(cuisineTopics)
```

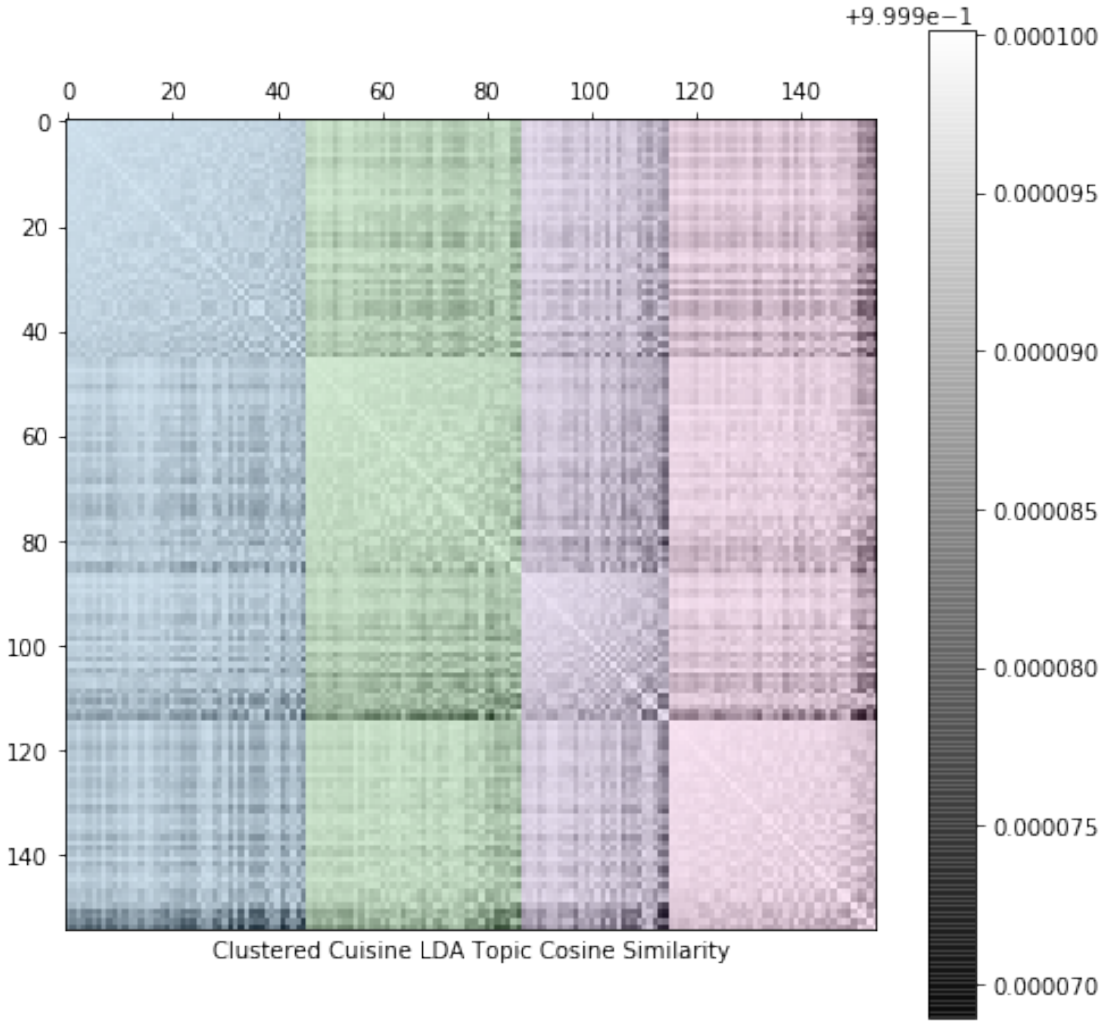
In [38]:
```
# Add cluster labels and score to cuisines dataframe
cuisines["cluster_km"] = cuisineTopicClustersKM.labels_
```

In [39]:
```
# Calculate distance to cluster centroid
getCentroidDistance = lambda r: paired_cosine_distances(\
        cuisineTopics.getrow(r.name).todense(), \
        cuisineTopicClustersKM.cluster_centers_[r["cluster_km"]].reshape(1, NUM_TOPICS)
)[0]
cuisines["centroid_dist_km"] = cuisines.apply(getCentroidDistance, axis=1)
```

In [40]:
```
# Create sorted index by cluster ID and distance to
# cluster centriod
sortedIdx = cuisines.sort_values(by=["cluster_km", "centroid_dist_km"]).index.values
```

In [41]:
```
# Get categorical color map for overlay
cmap_tab10 = plot.get_cmap("tab10")
```

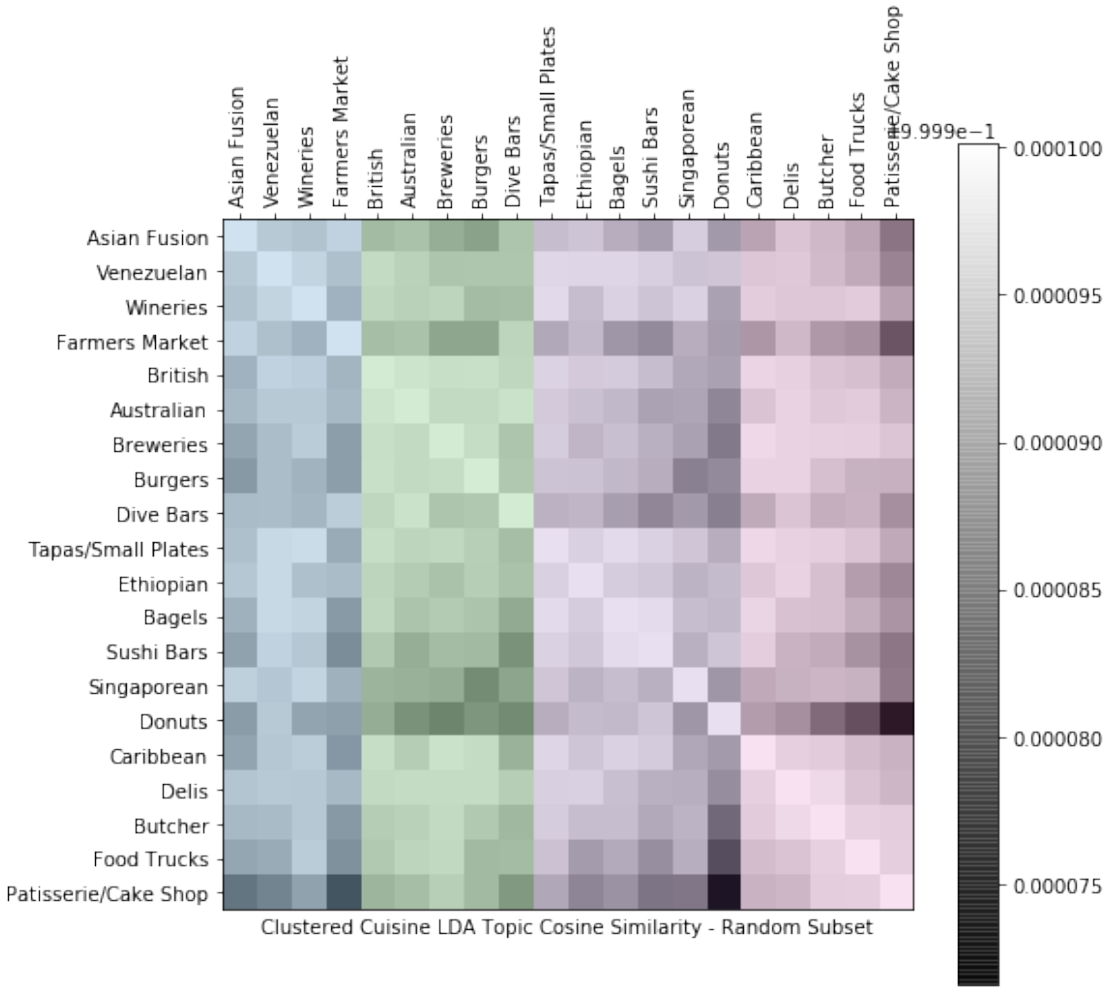
In [42]:
```
# Plot cosine similarity for all clustered cuisine
# LDA topics
matrix = cosine_similarity(cuisineTopics[sortedIdx])
labels = cuisines.cuisine[sortedIdx]
clusterCols = np.array([cuisines.cluster_km[sortedIdx].values / 5, ] * cuisines.shape[0
_, _ = graphSimilarity(matrix, labels, \
                       "Clustered Cuisine LDA Topic Cosine Similarity", \
                       display_labels=False, colors=cmap_tab10, \
                       matrix_under=clusterCols, alpha_over=0.8)

# Show plot
plot.show()
```



Each vertical band of color represents a cluster of similar cuisines. The resulting squares along the diagonal highlight the most similar cuisines pushing disimilarities toward the edges of the graph.

```
In [43]: # Sort sample
         SORTED_SAMPLE_20 = [i for i in sortedIdx if i in SAMPLE_20]

In [44]: # Plot cosine similarity for random 20 clustered cuisine
         # LDA topics
         matrix = cosine_similarity(cuisineTopics[SORTED_SAMPLE_20, :])
         labels = cuisines.cuisine[SORTED_SAMPLE_20]
         clusterCols = np.array([cuisines.cluster_km[SORTED_SAMPLE_20].values / 5, ] * 20)
         _, ax = graphSimilarity(matrix, labels, \
                                 "Clustered Cuisine LDA Topic Cosine Similarity - Random Subset"
                                 display_labels=True, colors=cmap_tab10, \
                                 matrix_under=clusterCols, alpha_over=0.8)
```

```
# Show plot
plot.show()
```



Zooming in shows the value of clustering. Cuisines with strong similarities in the real world appear with each other. This makes comparison between them visually easier. Clustering also reveals a weakness in using review text to represent cuisines: It includes significant non-cuisine information. The {Asian Fusion, Dive Bars, Tapas/Small Plates, Brewies} cluster more likely reflects 'trendy dining' than it does cuisine.