# Ingredient Substitution And Filtering Using Natural Language Processing Techniques

**Author:** Samuel Bowen, C1931367
**Project supervisor:** Jose Camacho Collados
**Module code:** CM3203
**Number of credits:** 40

## Table of Contents

# Introduction

In 2018, approximately seven-million people in the United Kingdom were following a meat free diet (vegetarian, pescaterian and vegan diets), and that number is expected to increase significantly in the coming years[1]. As the number of people following meatless diets increases, it will become important to adapt traditionally carnivorous recipes into meatless recipes by substituting ingredients.

Making international recipes can be difficult for several reasons. International ingredients can be expensive due to travel costs, and can be difficult to source. Ingredient substitution can make international recipes more accessible while minimally altering their flavour profile.

One way of generating ingredient substitutions is to train word embeddings on a large corpus of textual data. While several studies have used word embeddings to generate ingredient substitutes[2][3], no studies have used the word embedding vectors to train a classifier to label ingredients by dietary classification (vegan, vegetarian, pescatarian, and carnivore). By training a classifier to label ingredients by dietary classification, ingredient substitutes could be labelled and subsequently filtered upon generation.

In this project, I intend to explore the recipe-1m+ dataset using Word2Vec, FastText, and BERT word embeddings, creating a program that can suggest generally applicable food substitutions and can filter its results by dietary restriction using a classifier.

# Background

## Word Embeddings and Classifiers

A word embedding is a learned representation for text, in the form of a vector, that encodes the meaning of a word. Word embeddings close to one another in dimensional space share a similar meaning [4]. By training word embeddings on a corpus of recipes (such as the *recipes-1m+* dataset), ingredients near to one another in dimensional space should have a similar meaning, and thus may substitute for one another.

There are many word embedding models, each with their own advantages: Word2Vec uses a neural network to give each distinct word a vector [5], using words to predict words. FastText splits words into n-grams (sets of co-occurring characters), with words being represented by a sum of their n-grams. This allows FastText word embeddings to better represent morphologically rich language [6]. BERT, unlike Word2Vec or FastText, generates context-dependent embeddings. One word can have multiple vector representations [7].

Several papers have shown that natural language processing techniques (primarily word embeddings) can be used to suggest generally applicable ingredient substitutes. One paper uses the Word2Vec and BERT algorithms trained on the 1m+ recipe dataset[2] to produce context-free ingredient substitutions by comparing ingredient similarity [3]. Another paper uses a skip-gram model with negative sampling and compares ingredient similarity [8].

Classifiers are machine learning algorithms used to assign a class label to a given input. Supervised-learning classifiers use labelled training data, consisting of a number of features (measurable characteristics of the training data) [9].

### Research Questions

In order to demonstrate achievement of my stated aim, it will be necessary to clean and normalise the recipe-1m+ dataset; train BERT, Word2Vec and FastText word embeddings to suggest generally applicable ingredient substitutions; scrape a labelled dataset from the internet that can be used to train an ingredient classifier; clean and normalise the scraped dataset; train a classifier using the scraped dataset that can be used to accurately classify all ingredients within the dataset; and report on the performance of the trained word embedding and classifier models.

# Methodology

The aim of this project is to create a program that can suggest ingredient substitutions, and filter ingredients by their dietary classification. This can be broken down into two problems: training word embeddings, and training a classifier.

The following methodology will be employed to train Word2Vec, FastText, and BERT word embeddings:

1. Analyse the recipe-1m+ dataset.
2. Clean and standardise the recipes-1m+ dataset.
3. Train the Word2Vec, FastText, and BERT word embeddings.
4. Evaluate the Word2Vec, FastText, and BERT word embeddings.
5. Optimise the hyperparameters of the word embedding models to reduce loss.
6. Repeat steps 3 to 5 until optimised models have been created.

A dataset will be created from the word embeddings of the *recipes-1m+* dataset, taking each word's vector as features, and labelling each ingredient one of four dietary categories: vegan, vegetarian, pescatarian, and carnivore. If a classifier can be trained to accurately label ingredients by their dietary classification, then ingredient substitutions could be labelled by a classifier upon generation and subsequently filtered.

The following six classifiers have been selected for their availability in numerous machine learning libraries and efficiency: k-nearest-neighbour, random forest, gradient boosting, XG-boosting, support vector machines, and neural networks.

The following methodology will be employed to train the ingredient classifiers:

1. Analyse the ingredients of the *recipe-1m+* dataset.
2. Clean and standardise the ingredients of the *recipe-1m+* dataset.
3. Label the ingredients of the *recipe-1m+* dataset.
4. Split the dataset into two sets, the training set and the test set, using an 80/20 split. It is important the same training/test split is used to train all models, as different training/test splits may cause variations in accuracy.
5. Train the classifier models using the training set.

6. Evaluate the classifier models using the test set; confusion matrices could be generated to better visualise the relationship between the true and predicted labels of all classes.
7. Optimise the hyperparameters of the classifier models to increase accuracy; grid search and random search can be used to effectively search the hyperparameter space.
8. Repeat steps 5 to 7 until optimised models have been created.

Another dataset, such as a dataset containing the images of all the ingredients in the *recipes-1m+* dataset, could be generated using image scraping. Images could be used to train a neural network classifier.

The following methodology will be employed to train an image classifier:

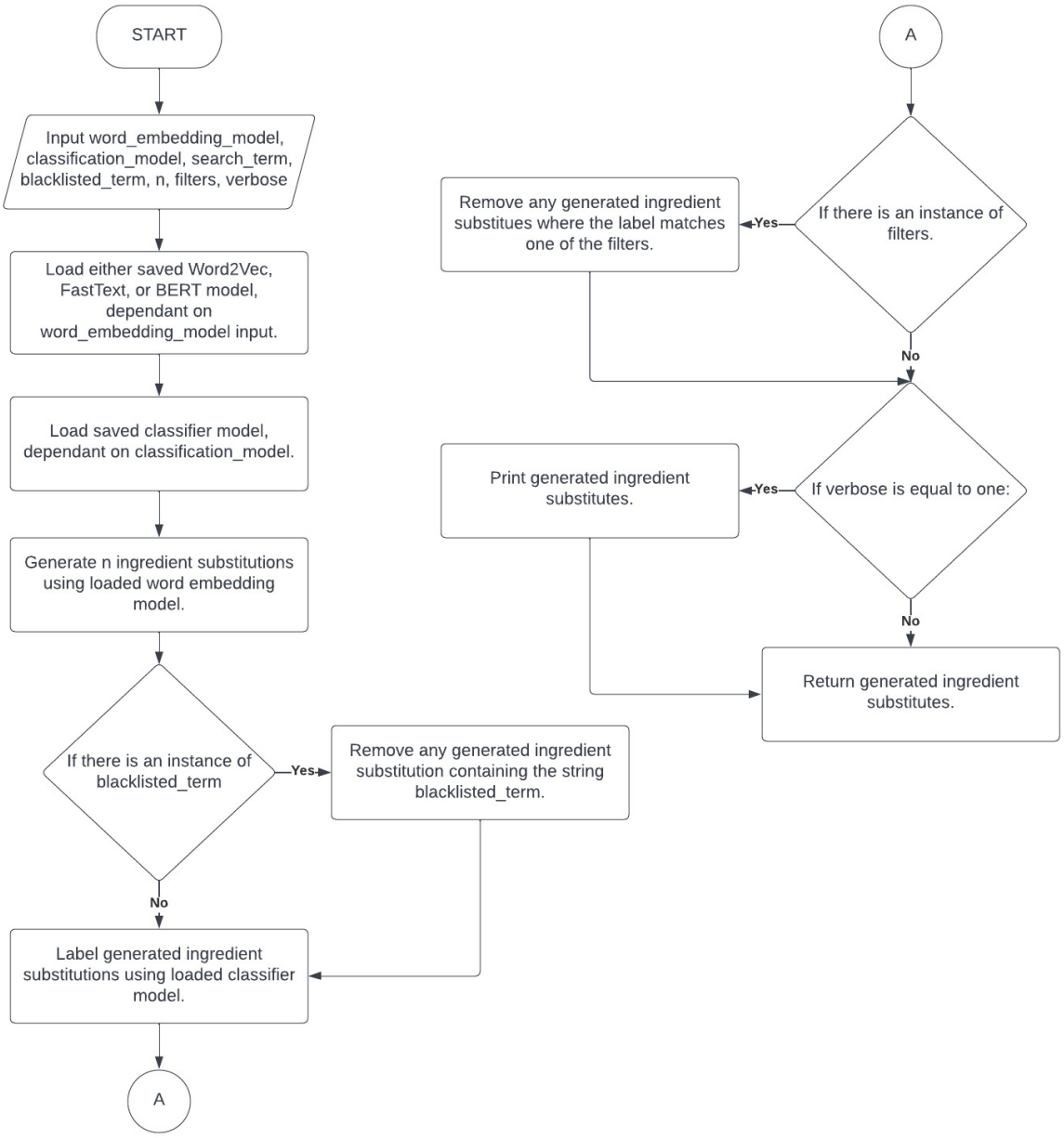
1. Use image scraping to create an image dataset.
2. Use the labels from the ingredient classifier to label the images.
3. Split the dataset into two sets, the training set and the test set, using an 80/20 split. It is important the same training/test split is used to train all models, as different training/test splits may cause variations in accuracy.
4. Train the image classifier model using the training set.
5. Evaluate the classifier models using the test set; confusion matrices could be generated to better visualise the relationship between the true and predicted labels of all classes.
6. Optimise the hyperparameters of the classifier models to increase accuracy; grid search and random search can be used to effectively search the hyperparameter space.
7. Repeat steps 5 to 7 until optimised models have been created.

The word embeddings and an ingredient classifier could be combined to create a function that generates ingredient substitutes, labels the ingredient substitutes according to dietary classification, and filters labelled ingredients accordinging to dietary restriction. The function would take the following inputs:

| **Input** | **Type** | **Valid inputs** | **Explanation** |
|---|---|---|---|
| word_embedding_model | String | “word2vec”<br>“fasttext”<br>“bert” | This input is used to select which word embeddings will be used to generate ingredient substitutions. |
| classification_model | String | “k_nearest_neighbour”<br>“random_forest”<br>“gradient_boosting”<br>“xg_boosting”<br>“support_vector_machine”<br>“neural_network” | This input is used to select which classification model labels the generated ingredient substitutes. |
| search_term | String | For Word2Vec and BERT | This input specifies the |

| | | | |
|---|---|---|---|
| | | word embeddings, any word that already has a pre-trained word embedding. For FastText word embeddings, any string. | ingredient to generate substitutes of. |
| blacklisted_term | String | Any string. | Any generated ingredient substitute containing this string will be removed. This input has been included to filter out substitutes that are too similar to the search term to be helpful. |
| n | Int | Any integer greater than or equal to 0. | This input specifies the number of ingredient substitutes to generate, before filtering. |
| filters | List of Strings | A list containing any of the following four strings:<br>“vegan”<br>“vegetarian”<br>“pescatarian”<br>“carnivore” | This input specifies which dietary classifications to filter out. |
| verbose | Int | The integers 0 and 1. | This input specifies whether the results of the function should be printed to the console. |

Find a flow chart of the function below:




# Implementation

## Word Embeddings

The Word2vec and FastText models were trained on the *simplified-recipes-1m+* dataset. This dataset is provided in the .npz format, and contains two separate numpy arrays: recipes and ingredients. The recipes numpy array contains approximately 1m recipes, containing on average 17 ingredients per recipe [10]. Every recipe in the recipes numpy array consists of a list of ingredients, and each recipe is stored as a list of indices. The ingredients numpy array contains 3500 ingredient-index pairs, ordered by their frequency in the recipes numpy array.

This dataset came pre-cleaned, though many artefacts from the cleaning process (such as the red in red pepper, or the blue in blue crabs, ECT.) had to be manually removed from the dataset. During this process, 759 ingredients were removed from the ingredients numpy array, leaving a total of 2741 ingredients.

The Word2Vec and FastText models were trained using the Python module, Gensim. Gensim is the fastest library for training vector embeddings, is well-documented, and provides robust, well-tested algorithms [11]. The initial Word2Vec and FastText models performed poorly, suggesting ingredient substitutes that had no relevance. This was fixed during hyperparameter tuning, when the window size of the models was changed from the default value of 3 to 20. As the ingredients of each recipe are unordered, for an ingredient to be given a proper embedding, the window size must be large enough for it to capture all other ingredients within the recipe.

The values returned by the loss functions when training the Word2Vec and FastText modules were ludicrously high. This is a known bug in Gensim version 3.6. I was unable to update my version of Gensim, as the model.wv.vocab function, which I was using to analyse the generated vectors, is deprecated in all later versions. I was confused by this bug for some time, until I found a workaround [12]. See the code below:

```
class LossLogger(CallbackAny2Vec):
    '''Output loss at each epoch'''
    def __init__(self):
        self.epoch = 0
        self.loss_previous_step = 0

    def on_epoch_end(self, model):
        loss = model.get_latest_training_loss()
        if self.epoch == 0:
            print('Loss after epoch {}: {}'.format(self.epoch, loss))
        else:
            print('Loss after epoch {}: {}'.format(self.epoch, loss-
self.loss_previous_step))
        self.epoch += 1
        self.loss_previous_step = loss

loss_logger = LossLogger()
```

This loss logger keeps track of the current epoch, and calculates the difference between the current loss value and the previous loss value. So long as the difference between the loss of each epoch is decreasing, the model is improving. Both the Word2Vec and FastText models were trained over twenty epochs. The loss had begun to plateau at twenty epochs, with any additional epochs reducing loss minimally.

The Word2Vec and FastText models were trained with the following hyperparameters:

```
model_W2V = Word2Vec(
    recipe_list,
    window=20,
    size=100,
    min_count=1,
    iter=20,
    callbacks=[loss_logger],
    compute_loss=True,
    workers=4)
```

```
model_FastText = FastText(
    recipe_list,
    window=20,
    size=100,
    min_count=1,
    iter=20,
    callbacks=[loss_logger],
    compute_loss=True,
    workers=3)
```

The three most important hyperparameters for training Word2Vec and FastText word embeddings were window (the size of the window), iter (the number of epochs), and size (the number of dimensions the word embedding vectors are given).

The only difference in hyperparameters between the two models is the number of workers used in training the FastText model. When using all four of my CPUs cores to train the FastText model, my Python would crash from CPU overuse.

Dimensional reduction algorithms, such as T-SNE[13], were used to reduce the size of the word embedding vectors to two dimensions, so that they could be plotted onto a 2D graph. To achieve this, the vectors of each model's word embeddings were used to train a T-SNE model, provided by the Python module sklearn. See the code, and an example graph in figure-1:

```
vocab = list(model_FastText.wv.vocab)
x = model_FastText.wv[vocab]

tsne = TSNE(n_components=2)
x_tsne = tsne.fit_transform(x)

df = pd.DataFrame(x_tsne, index=vocab, columns=['x', 'y'])

fig = plt.figure(figsize=(15,15), dpi=80)
ax = fig.add_subplot(1, 1, 1)

first_fifty = df.head(50)
ax.scatter(first_fifty ['x'], first_fifty ['y'])

for word, pos in first_fifty .iterrows():
    ax.annotate(word, pos)

plt.show()
```

## Ingredient Classifier

To train the ingredient classifier, I first had to label the 2741 ingredients in the ingredients numpy array. Attempts were made to use the Python module Selenium and the doublecheckvegan website [14] to automatically label the dataset, but the doublecheckvegan website proved to be too limited. Because of this, I was forced to manually label every ingredient within the dataset, which took a significant amount of time. I encountered some difficulties labelling the vegan and carnivore ingredients within the dataset. Some wine manufacturers use egg white and milk powder as fining agents in wine, and some do not. Some cheese manufacturers use animal rennet to create cheese curds, white some use vegetable rennet. As the ingredients within the dataset almost never give a corresponding brand, it is impossible to know whether the given ingredient is vegan or carnivorous. In such instances, I have chosen a dietary classification and stuck with it. For example, all generic cheeses within the dataset are labelled as carnivorous.

After the ingredients were labelled, the vectors of the Word2Vec and FastText word embedding models were added to a CSV file containing all of the labelled ingredients. This resulted in the creation of two datasets:
*ingredient_dataset_100_dimensions_word2vec.csv* and
*ingredient_dataset_100_dimensions_fasttext.csv*.

The python module sklearn was used to train all ingredient classifier models. Sklearn was selected for its ease of use, analytical functions, convenient visualisation functions, and thorough documentation [15].

These datasets were both split into a test/train split using sklearn's *train_test_split* function, and were used to train six different types of image classifier: k-nearest-neighbour, random forest, gradient boosting, XG-boosting, support vector machines, and neural networks. Some image classifiers that were quick to train, such as gradient boosting and k-nearest-neighbour, had their hyperparameters tuned using a grid search. Other classifiers that were slow to train, such neural networks, had their hyperparameters manually optimised, as it would have taken too long to search the hyperparameter space, especially with cross validation. See example code for hyperparameter optimisation below:

```
gradient_boosting_model = ensemble.GradientBoostingClassifier(verbose =
0)

grid_list = {"n_estimators": [10, 25, 50, 100, 250, 500, 750, 1000],
             "max_depth": [1, 2, 3, 5, 10, 20, 50]
             }

grid_search = GridSearchCV(gradient_boosting_model,
                           param_grid = grid_list,
                           verbose = 4,
                           scoring = "accuracy",
                           n_jobs = 4,
                           cv = 3)

grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
```

After optimising the hyperparameters of several classifiers, it was obvious that classifiers trained on the FastText dataset performed better than the classifiers trained on the Word2Vec dataset. A second Word2Vec dataset was created, where the word embeddings were trained to 50 epochs instead of the original 20. Several classifiers were trained on this new Word2Vec dataset, but no performance increase could be identified. From this point forward, I exclusively trained classifiers on the FastText dataset.

After all classifiers were trained and optimised on the *ingredient_dataset_100_dimensions_fasttext.csv* dataset, two more datasets were created by training two more FastText word embedding models with size hyperparameters of 50 and 150 respectively. These datasets were called *ingredient_dataset_50_dimensions_fasttext.csv, and ingredient_dataset_150_dimensions_fasttext.csv.* Classifier models were trained using both datasets.

A dataset was created by using T-SNE dimensional reduction algorithm to turn a 100 dimensional FastText model into a 50 dimensional FastText model. This dataset, however, trained worse performing classifiers than *ingredient_dataset_50_dimensions_fasttext.csv,* and so no more classifiers were trained using this dataset.

The performance of all classifiers was assessed using sklearn's *classification_report* and *confusion_matrix* functions. Sklearn's *classification_report* function takes a model's predicted labels, and the true labels of the dataset, and prints a comprehensive report that describes the systems accuracy, precision, recall, and f1-score. Sklearn's confusion matrix takes the same arguments, and generates a confusion matrix. The python module matplotlib (and the function DisplayConfusionMatrix), were used to create plots of the confusion matrices. See code for performance analysis of a support vector machine model below:

```
filename = 'svm_classifier_fasttext_50.sav'
svm_classifier = pickle.load(open(filename, 'rb'))

y_pred = svm_classifier.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=["vegan", "veg", "pes", "car"])
disp.plot()
plt.show()

print(classification_report(y_test, y_pred, target_names=["vegan", "veg",
"pes", "car"]))
```

In classifiers where it was available, the class_weight hyperparameter was set to balanced, which helped to address the class imbalance of the dataset.

## Image Classifier

To train the image classifier, I first had to scrape 2741 images from the internet. The Python module Selenium was used to scrape the first image result on Google Images for each ingredient within the ingredients dataset. I encountered several problems while image scraping. Firstly, the code would sometimes stop working when attempting to save an image of cornflakes. I was unable to fix this bug, and there are several ingredients within the dataset where the first Google Image result is cornflakes. This error likely occurred due to a corrupted file, or an unanticipated image format that my program was unable to save. Another issue was the ambiguity of some ingredient's names. When searching for some ingredients (especially generic ingredients like oil, fish, ECT.), the first result on Google Images was not an ingredient (for example, a barrel of oil, or a fish in the ocean). This could have been fixed by appending the word ingredient onto the end of each query. After all of the images were scraped, I spent a significant amount of time looking through the dataset, removing inappropriate images and replacing them with appropriate images.

The python module Tensorflow was selected for its fast deployment, thorough documentation, and its GPU support, which can be used to increase model training speed by up to six times [16].

After creating the image dataset, I split the images into a test/train split. The test/train split was used to train two image classification models: a simple image classifier, and an image classifier that uses transfer learning.

Keras' ImageDataGenerator function was used to produce more training input from the limited dataset available. This function takes the input images and several parameters, and then applies several transformations to them to create more novel input for the neural network.

For a significant period of time, the trained image classifiers performed poorly, achieving 0.69 accuracy by classifying all ingredients as vegan. This issue was fixed by passing class weights to the class_weights hyperparameter of the model. Class weights were manually calculated using the following formula[17]:

```
(1/num_of_class_samples) * (total/2)
```

After manually specifying the class_weights hyperparameter, the image classifier began to correctly predict the labels of some ingredients that were not vegan, but the overall accuracy was significantly lower. The simple image classifier used the following structure:

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu',
input_shape=(image_height,image_width, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(96, activation='relu'),
    Dropout(0.5),
```

```
    Dense(4, activation='softmax')
])

opt = SGD(lr=0.00001)
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics =['accuracy'])
```

The model uses the ReLU activation function, the most commonly used activation function in neural networks. The ReLU activation was selected because it is computationally efficient and prevents vanishing gradients. Several dropout layers have been used throughout the model. Dropout layers randomly drop a percentage of neurons at every step during training time, which helps to prevent overfitting. A dropout layer with a value of 0.5 has been used before every dense layer in the network[18]. The final dense layer creates the output for the image classifier. The softmax activation function normalises the output of a neural network to a probability function over the distributed classes, effectively creating confidence levels that a given image belongs to a given class.

A second image classifier model was created using transfer learning. To create a transfer learning model, the weights of the pretrained classifier Imagenet were loaded, the first fifteen layers of the Imagenet classifier were locked, so that the weights would not change during training, and a small model was appended to the end. The transfer learning image classifier used the following structure:

```
ResNet_model = tf.keras.applications.ResNet152V2(weights='imagenet',
include_top=False, input_shape=(img_h, img_w, 3))

for layer in ResNet_model.layers[:-15]:
    layer.trainable = False

x = ResNet_model.output
x = GlobalAveragePooling2D()(x)
x = Flatten()(x)
x = Dense(units=512, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(units=512, activation='relu')(x)
x = Dropout(0.3)(x)
output  = Dense(units=4, activation='softmax')(x)
model = Model(ResNet_model.input, output)
```

Initially, the values for the dropout layers were 0.5, but this reduced the accuracy of the model, and so the values were lowered to 0.3.

The image classifiers were tested using sklearn's classification_test and confusion matrix functions, in an identical way to the ingredient classifiers.

## Ingredient Filter Function

The ingredient filter function takes the following inputs: substituter, classifier, search_term, blacklisted_term, n, filters, and verbose. The function first validates each of the inputs, using the python isinstance function to ensure each input exists and is the correct data type. Some validation, such as the filters validation, uses sets and the python issubset function to ensure only the ingredient classes can be entered as filters. Find the filters validation code below:

```
filter_testing = ["vegan", "vegetarian", "pescatarian", "carnivore"]
        filter_testing = set(filter_testing)
        input_filter_testing = set(filters)
        if not(input_filter_testing.issubset(filter_testing)):
            print("filters must be a list containing only the following
strings: 'vegan', 'vegetarian', 'pescatarian' and 'carnivore'.")
            return "error"
```

After the inputs are validated, the word embedding model and the classifier model are loaded using the Python pickle module and the Gensim load function. The ingredient_dataset_100_dimensions_fasttext.csv dataset is also loaded into a Pandas dataframe. A number of substitutes equal to n are generated and stored in a list of tuples, and any substitutes containing the blacklisted_term are removed. At some point during development, the ingredients used to train the word embedding models and the classifiers became slightly misaligned. Some ingredients suggested by the word embedding model do not have corresponding vectors in the ingredient_dataset_100_dimensions_fasttext.csv dataset (such as casings, an artefact of cleaning the dataset that must have been spotted and removed after the word embedding models were already trained). The following code is used to catch these ingredients and delete them from the ingredient substitutions:

```
if bool(row):
       ingredient_substitutions.append(ingredient)
       ingredient_similarity.append(similarity)
       ingredient_substitutions_dimensions.append(row[0])

else:
       error_catching.append(i)
```

If row, taken from the ingredient_dataset_100_dimensions_fasttext.csv dataset, does not exist, then add the ingredient to an error catching list to be deleted.

All the word embedding vectors of the substituted ingredients are then given to the classier, which predicts the label of each ingredient. If an ingredient is given a label that matches one of the classes in the filters input, it is removed. The output, a pandas dataframe, is then returned.

I was unable to implement BERT word embeddings. BERT took significantly longer to train when compared to the Word2Vec and FastText word embedding algorithms, and frequently crashed during training, no matter how I adjusted the training parameters. I left implementing BERT word embeddings far too late into the development process, and did

not have enough time to figure out how to pool layers into a format that could be used generating ingredient substitutes or for training classifiers.

# Results and Evaluation

## Ingredient Classifier

The performance of every classifier has been tested using the *classification_report* function provided by sklearn. The *classification_report* function provides a detailed analysis of a classifier, taking the predicted and true labels of a given dataset, and outputting the precision, recall, and f1-score. Every classifier has also been tested using a confusion matrix, using the *confusion_matrix* function provided by sklearn. Every confusion matrix was manually analysed to ensure that the classification report was accurate. Several classifiers had high precision, but miscategorised large portions of the dataset. Confusion matrices were a quick, visual way to verify whether this was happening.

All classifiers tested were trained on FastText word embeddings. During the preliminary training of the classifier models, it was identified that classifiers trained on Word2Vec word embeddings always performed worse than classifiers trained on FastText word embeddings, even after hyperparameter tuning.

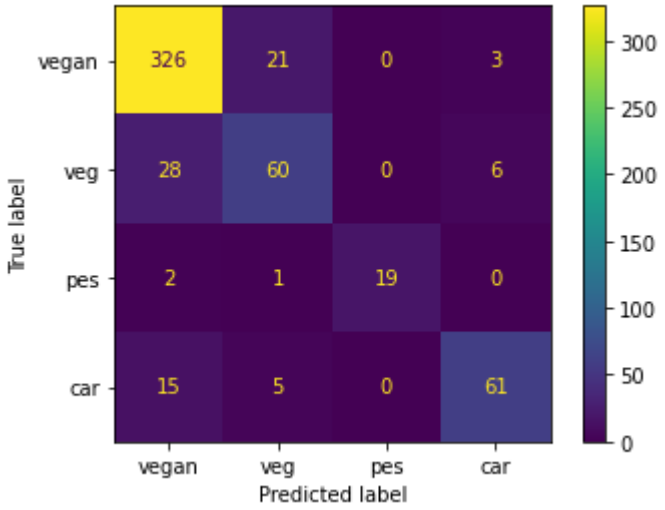
Find a table created using the *classification_report* function below:

| **Classifier Evaluation Table** | | | | |
|---|---|---|---|---|
| **Classifier** | **Accuracy** | **Precision (Macro Average)** | **Recall (Macro Average)** | **F1 Score (Macro Average)** |
| Gradient Boosting, 50 dimensions | 0.81 | 0.82 | 0.69 | 0.74 |
| Gradient Boosting, 100 dimensions | 0.83 | 0.85 | 0.71 | 0.76 |
| Gradient Boosting, 150 dimensions | 0.82 | 0.84 | 0.72 | 0.77 |
| K-nearest-neighbour, 50 dimensions | 0.85 | 0.83 | 0.80 | 0.81 |
| K-nearest-neighbour, 100 dimensions | 0.85 | 0.88 | 0.78 | 0.82 |
| K-nearest-neighbour, 150 dimensions | 0.84 | 0.84 | 0.78 | 0.81 |
| Neural Network, 50 dimensions | 0.84 | 0.80 | 0.79 | 0.79 |
| Neural Network, 100 dimensions | 0.84 | 0.83 | 0.79 | 0.81 |

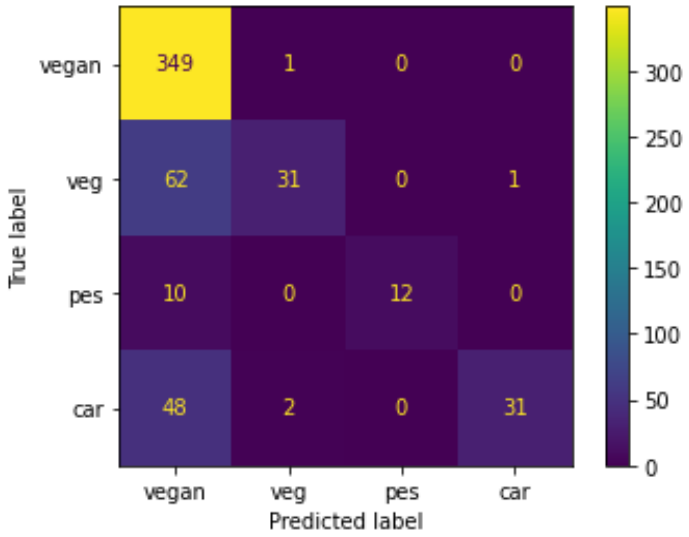
| | | | | |
|---|---|---|---|---|
| Neural Network, 150 dimensions | 0.83 | 0.82 | 0.77 | 0.79 |
| Random Forest, 50 dimensions | 0.76 | 0.87 | 0.60 | 0.69 |
| Random Forest, 100 dimensions | 0.79 | 0.89 | 0.60 | 0.69 |
| Random Forest, 150 dimensions | 0.77 | 0.91 | 0.56 | 0.65 |
| Support Vector Machine, 50 dimensions | 0.81 | 0.77 | 0.78 | 0.78 |
| Support Vector Machine, 100 dimensions | 0.83 | 0.83 | 0.80 | 0.81 |
| Support Vector Machine, 150 dimensions | 0.85 | 0.86 | 0.80 | 0.83 |
| XG-Boosting, 100 dimensions | 0.82 | 0.86 | 0.68 | 0.75 |
| XG-Boosting, 150 dimensions | 0.82 | 0.85 | 0.73 | 0.77 |
| Image Classifier | 0.56 | 0.50 | 0.54 | 0.52 |
| Image Classifier, Transfer Learning | 0.62 | 0.54 | 0.65 | 0.55 |

The tests above show that classifiers, trained on the dimensional features of word embeddings, can be used to accurately classify ingredients into different dietary categories. The most successful models were the k-nearest-neighbour and the support vector machine models. The KNN (k-nearest-neighbour) model and the SVM model have the same accuracy of 0.85. The KNN model has a higher precision of 0.88, but the SVM model has a  higher recall and f1-score.

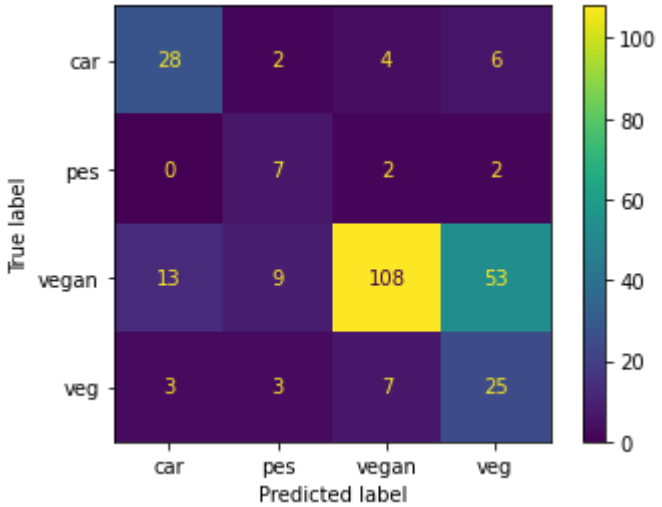
The SVM is the best performing model, achieving a good compromise between precision, recall and f1-score. The confusion matrix of the SVM 150 dimensional model can be found below:



While the random forest models had the highest precision, they had some of the lowest accuracy, recall and f1 scores. This is because the random forest model frequently miscategoried ingredients as vegan, the most common class in the dataset. The confusion matrix of the random forest 150 dimensional model can be found in below:




The image classifiers were not able to accurately classify ingredients by dietary restriction. The two image classifier models, like the decision tree models, miscategorised a large number of ingredients as vegan. Without specifying class weights, the image classifiers categorised every ingredient as vegan, which suggests they had not learnt from the images at all. Even after specifying class weights, the image classifiers only improved marginally, frequently miscategorising vegan ingredients as ingredients from other classes. The confusion matrix for the transfer learning image classifier can be found below:




This is likely because images of ingredients do not provide enough data to make an accurate classification. Some ingredients are easily misidentified, such as flour and gelatin powder (a

vegan and a carnivorous ingredient respectively, both with the appearance of white powder), ECT.. The classifier could not effectively distinguish between these ingredients.

## Word Embeddings

To assess the performance of the Word2Vec and FastText word embedding models, a ground truth evaluation table had to be created. 15 ingredients were selected, with 10 ingredients selected from the 500 most common ingredients, and 5 ingredients selected from the remaining 2241 ingredients. A list of ingredient substitutes for each ingredient was created using the Food Substitutions Bible [19]. Any ingredient substitutions suggested by the Food Substitution Bible that were not in the ingredients dataset were omitted. Additional ingredient substitutions were added according to my personal intuition and several online recipe blogs [20][21][22]. These online sources are less reliable than the Food Substitutions Bible, but suggested more ingredient substitutes that were within the ingredients dataset, and were necessary to create a more thorough ground truth table. The ground truth evaluation table can be found in figure 3.

To assess the accuracy of the Word2Vec and FastText word embedding models, the following steps were taken:

1. Every ingredient in the ground truth table is run through the ingredient_filter function (where search_term is the name of the ingredient and n is equal to 10)to create a list of ingredient substitutes.
2. The list of ingredient substitutes is compared to the ground truth table, and the number of correct predictions are counted.
3. The accuracy of each individual ingredient is calculated by the formula: total number of ingredient substitutions / number of correct ingredient substitutions.
4. A total accuracy is calculated by averaging each individual accuracy.

See the following tables for the accuracy of the word embedding models:

| **FastText Ingredient Substitution Analysis, n = 20** | | | |
|---|---|---|---|
| **Ingredient** | **Number Potential Substitutions** | **Number of Correct Substitutions** | **Accuracy** |
| Bacon Slices | 7 | 2 | 0.28 |
| Cream | 5 | 2 | 0.40 |
| Olive oil | 8 | 4 | 0.50 |
| Corn Syrup | 7 | 4 | 0.57 |
| Ricotta cheese | 9 | 3 | 0.33 |
| Brown lentils | 9 | 4 | 0.44 |
| Chicken breast | 3 | 0 | 0.00 |

| | | | |
|---|---|---|---|
| Green onion | 7 | 4 | 0.57 |
| Noodles | 4 | 2 | 0.50 |
| Potato | 5 | 3 | 0.60 |
| Smoked Salmon | 2 | 2 | 1.00 |
| Hazelnuts | 3 | 2 | 0.66 |
| Lamb neck | 4 | 2 | 0.50 |
| Pita bread | 5 | 3 | 0.60 |
| Coconut milk | 5 | 2 | 0.40 |
| | | **Average Accuracy** | 0.49 |

| **Word2Vec Ingredient Substitution Analysis, n = 20** | | | |
|---|---|---|---|
| **Ingredient** | **Number Potential Substitutions** | **Number of Correct Substitutions** | **Accuracy** |
| Bacon Slices | 7 | 3 | 0.42 |
| Cream | 5 | 2 | 0.40 |
| Olive oil | 8 | 2 | 0.25 |
| Corn Syrup | 7 | 4 | 0.57 |
| Ricotta cheese | 9 | 1 | 0.11 |
| Brown lentils | 9 | 4 | 0.44 |
| Chicken breast | 3 | 0 | 0.00 |
| Green onion | 7 | 3 | 0.42 |
| Noodles | 4 | 1 | 0.25 |
| Potato | 5 | 3 | 0.60 |
| Smoked Salmon | 2 | 2 | 1.00 |
| Hazelnuts | 3 | 2 | 0.66 |
| Lamb neck | 4 | 1 | 0.25 |
| Pita bread | 5 | 3 | 0.60 |
| Coconut milk | 5 | 2 | 0.40 |

| | |
|---|---|
| **Average Accuracy** | 0.42 |

The FastText model is 7% more accurate than the Word2Vec model, and performed equal to or better than the Word2Vec model on every ingredient but bacon slices, where the Word2Vec model was 14% more accurate. Both models were unable to generate correct ingredient substitutes for chicken breast, because all twenty generated ingredient substitutes were different cuts of chicken.

Both models have low accuracy scores, but I believe this is due to the difficulty of testing something as subjective as ingredient substitution. When creating the ground truth table, it was difficult to decide what ingredient substitutions should be included. Some ingredient substitutions aim to reduce the cost of a dish(for example, substituting saffron for annatto), some attempt to minimally modify a dishes flavour profile (for example, substituting cream for milk when no cream is available), and some attempt to modify a dish for a dietary need (a vegan might substitute chicken for mushrooms). The ground truth table is also too small, with only 15 ingredients. Ideally, the ground truth table would contain all 2,741 ingredients, but that would be infeasible to create within the time constraints of my project. If a larger truth table was used, then the accuracy of the Word2Vec and FastText models could be better measured.

## Ingredient Filter Function

The function, *ingredient_filter*, defined in ingredient_filter.py, combines the ingredient substituter and the classifier to create a function that can provide ingredient substitutes, and can filter them by dietary restriction. The function allows the user to select any classifier model, and either the Word2Vec or FastText word embeddings.

The tests table, found below, describes the tests done to ensure the function has proper input validation:

| **Test Type** | **Input** | **Expected Output** | **Actual Output** | **Pass/Fail** |
|---|---|---|---|---|
| Valid data | substituter = "fasttext"<br>classifier = "knn"<br>search_term = "onion"<br>blacklisted_term = "onion"<br>n = 100<br>filters = ["carnivore", "pescatarian"]<br>verbose = 1 | Ingredient substitutions list, with no pescatarian or carnivore ingredients | Ingredient substitutions list, with no pescatarian or carnivore ingredients | PASS |
| Invalid data | substituter = "bert" | bert is not a valid input. Try 'word2vec' or 'fasttext'. | bert is not a valid input. Try 'word2vec' or 'fasttext'. | PASS |
| Invalid type | substituter = 300 | 300 is not a valid input. Try 'word2vec' or 'fasttext'. | 300 is not a valid input. Try 'word2vec' or 'fasttext'. | PASS |

| | | | | |
|---|---|---|---|---|
| Null | substituter=None | None is not a valid input. Try 'word2vec' or 'fasttext'. | None is not a valid input. Try 'word2vec' or 'fasttext'. | PASS |
| Invalid data | classifier = “image_classifier” | image_classifier is not a valid input. Try 'gradient_boosting', 'knn', 'neural_network', 'random_forest', or 'svm'. | image_classifier is not a valid input. Try 'gradient_boosting', 'knn', 'neural_network', 'random_forest', or 'svm'. | PASS |
| Invalid type | classifier = -999 | -999 is not a valid input. Try 'gradient_boosting', 'knn', 'neural_network', 'random_forest', or 'svm'. | -999 is not a valid input. Try 'gradient_boosting', 'knn', 'neural_network', 'random_forest', or 'svm'. | PASS |
| Null | classifier=None | None is not a valid input. Try 'gradient_boosting', 'knn', 'neural_network', 'random_forest', or 'svm'. | None is not a valid input. Try 'gradient_boosting', 'knn', 'neural_network', 'random_forest', or 'svm'. | PASS |
| Invalid data | substituter = “word2vec” search_term=”rabbit shoulder” | search_term must be a word embedding. Check ingredient_dataset_100_dimensions_fasttext.csv for a full list of ingredients. | search_term must be a word embedding. Check ingredient_dataset_100_dimensions_fasttext.csv for a full list of ingredients. | PASS |
| Valid data | substituter = “fasttext” search_term=”rabbit shoulder” | Ingredient substitutions list | Ingredient substitutions list | PASS |
| Valid data | search_term=”pork” blacklisted_term=”pork” | Ingredient substitutions list, with none of the ingredients containing the string pork | Ingredient substitutions list, with none of the ingredients containing the string pork | PASS |
| Invalid type | search_term=888 | search_term must be a string. | search_term must be a string. | PASS |
| Invalid type | blacklisted_term = 782 | blacklisted_term must be a string. | blacklisted_term must be a string. | PASS |
| Null | blacklisted_term=None | blacklisted_term must be a string. | blacklisted_term must be a string. | PASS |
| Invalid type | n=”twenty” | n = twenty is an invalid input. Try inputting an integer greater than zero. | n = twenty is an invalid input. Try inputting an integer greater than zero. | PASS |
| Null | n=None | n = None is an invalid input. Try inputting an integer greater than zero. | n = None is an invalid input. Try inputting an integer greater than zero. | PASS |
| Extreme data | n=1000000 | Ingredient substitutions list, containing every | Ingredient substitutions list, containing every | PASS |

| | | | | |
|---|---|---|---|---|
| | | ingredient in the dataset | ingredient in the dataset | |
| Extreme data | filters=[“vegan”, “pescatarian”, “vegetarian”, “carnivore”] | Ingredient substitutions list, containing no ingredients | Ingredient substitutions list, containing no ingredients | PASS |
| Invalid type | filters = “vegan” | filters must be a list containing only the following strings: 'vegan', 'vegetarian', 'pescatarian' and 'carnivore'. | filters must be a list containing only the following strings: 'vegan', 'vegetarian', 'pescatarian' and 'carnivore'. | PASS |
| Invalid data | filters = [“vegan”, “vegetarian”, “keto”] | filters must be a list containing only the following strings: 'vegan', 'vegetarian', 'pescatarian' and 'carnivore'. | filters must be a list containing only the following strings: 'vegan', 'vegetarian', 'pescatarian' and 'carnivore'. | PASS |
| Valid data | filters=[] | Ingredient substitutions list, with no dietary restriction filter | Ingredient substitutions list, with no dietary restriction filter | PASS |
| None | filters=None | filters must be a list containing only the following strings: 'vegan', 'vegetarian', 'pescatarian' and 'carnivore'. | filters must be a list containing only the following strings: 'vegan', 'vegetarian', 'pescatarian' and 'carnivore'. | PASS |
| Valid data | verbose = 1 | Function will output pandas dataframe, and function will print pandas dataframe | Function will output pandas dataframe, and function will print pandas dataframe | PASS |
| Invalid type | Verbose=True | Function will output pandas dataframe, but will print nothing | Function will output pandas dataframe, but will print nothing | PASS |
| Extreme data | verbose = -9999999999 | Function will output pandas dataframe, but will print nothing | Function will output pandas dataframe, but will print nothing | PASS |

An example output of the function can be found in figure 2.

## Critical Appraisal of Project

Throughout the project, I employed a waterfall development approach[23], working in development sprints of three to five days to finish specific project deliverables. There were several periods of time where I was unable to work on the project, and so these intense periods of development helped me to make up for lost time.

I used the Python programming language for all programming tasks within the project. Python offers concise, readable code, with a wide variety of machine learning modules, and

is a leading language for data analysis[24]. I initially used Jupyter notebook for its ease of use, but found the repeated code and lack of debug features were restricting my progress. I later installed Anaconda, a python distribution platform for machine learning, and began programming in Spyder. Sypder's powerful debug features and variable explorer allowed me to work considerably faster.

I initially used Google Collab to train my word embeddings, but I found it to be too slow compared to the speed my local CPU could train the models. I later installed tensorflow-gpu to speed up the training of my image classifiers, which saved considerable time.

# Future Work

There are several elements of this project that could be developed in the future:

- BERT can be used to train contextualised word embeddings, which could be more precise than Word2Vec and FastText word embeddings[2].
- Image embeddings could be concatenated with word embeddings to potentially produce more accurate ingredient substitutions.
- BERT word embeddings could be used to train ingredient classifiers.
- The *simplified-recipes-1m+* dataset [10] was convenient as it came pre-processed, needed minimal cleaning, and a smaller dataset allowed for quicker model training. The full *recipes-1m+* dataset [25] would have the same number of recipes, but a significantly larger number of ingredients, which could allow for better ingredient substitution.
- Both images and word embeddings could be given to a neural network classifier as input. This would provide both models with more context, which could result in a model with higher accuracy.
- The performance of the SVM increased as the dimensionality of the dataset increased. SVMs are effective in high dimensional spaces], and so it is possible that an even higher dimensional space (such as a 200 dimensional dataset) could achieve better results.
- Better methods of assessing the performance of word embedding models are essential to creating better, more accurate ingredient substitutions.

# Conclusion

In this project, it is shown that Word2Vec and FastText word embedding models can be trained to adequately suggest ingredient substitutions, and that the FastText model performs best.

It is also shown that classifiers can be trained on word embedding vectors, which can accurately classify ingredients into four dietary categories (vegan, vegetarian, pescatarian and carnivore). A SVM trained on the word embeddings of the *ingredient_dataset_150_dimensions_fasttext.csv* dataset performed best at this task. It was

found that some classifier models perform better in higher dimensional (150 dimensions), and that no models perform better in lower dimensional space (50 dimensions).

t-SNE dimensionality reduction algorithms were used to create a novel dataset, which trained classifiers that were less accurate than the Word2Vec and FastText word embeddings.

An image classifier was also trained using a neural network and transfer learning, but was unable to classify ingredients accurately. Images do not hold enough information to classify images by dietary classification, as too many images (such as sugar and gelatin powder) look identical to one another while having different classifications.

The function, ingredient_filter, can suggest ingredient substitutions using trained Word2Vec and FastText models, can label ingredients (and subsequently filter them) using trained classifier models, and provides some advanced searching functions (the ability to blacklist a specific word, and create custom filters). The function has been tested, and has adequate input validation.

# Reflection

I have learnt many things throughout the course of this project. The importance of thoroughly analysing all datasets has been highlighted throughout this project. For example, had I more thoroughly analysed the simplified-recipes-1m+ dataset, I would have cleaned the recipes dataset sooner, saving me time in the long run. The importance of rigour, in training and comparing many different models, has been highlighted to me. Initially, I thought that neural networks would outperform all other classifiers. Had I not trained classifier models on all of the various datasets (50 dimensions to 150 dimensions), and thoroughly compared the performance of each, this bias may have been carried through the study. The importance of cleaning datasets has been highlighted to me. The adage ‘garbage in, garbage out” is true, especially for my image classifier, which I suspect had poor performance due to the unsuitability of the input and the need for better, cleaner data. In future projects, I will ensure that I use better image scraping methods in future projects, which are more robust and provide cleaner data. I have also learnt the importance of time management. No individual word embedding model or classifier took longer than a few hours to train, but that time can quickly add up, especially when comparing so many classifier models. In my initial plan, time was allocated for learning the theory behind word embedding algorithms, classifiers, ECT.. In the future, I believe it would be more productive to jump straight into programming, learning the theory as and when needed, in order to create a minimum viable product faster.

# Acknowledgements

This project was supported by Jose Camacho Collados, whose insights and suggestions were invaluable to the completion of the project. Thank you to Emma Pead, my partner, who has supported me unconditionally throughout the duration of my project.

# References


[1] Benson, A.; Irdam, D.; Bulceag, I. and Barber, T. 2019. *The Food and You Survey.* London: Food Standards Agency and NatCen.

[2] Marín, J. *Et al.* 2019. Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. DOI: 10.1109/TPAMI.2019.2927476

[3]Pellegrini, C.; Özsoy, E.; Wintergerst, M. and Groh, G. 2021. Exploiting Food Embeddings for Ingredient Substitution. *Proceedings of the 14th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2021)*. Vienna, Austria 11-13 February 2021. Setúbal, Portugal: SCITEPRESS. DOI: 10.5220/0010202000670077

[4] Jurafsky, D. and James, M. H. 2021. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. 3rd Edition draft. Upper Saddle River, N.J.: Prentice Hall. ISBN 978-0-13-095069-7.

[5] Mikolov, T.; Chen, K.; Corrado, G. and Dean, J. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv.* arXiv:1301.3781. Version 3. DOI: https://doi.org/10.48550/arXiv.1301.3781

[6] Bojanowski, P.; Grave, E.; Joulin, A. and Mikolov,T. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics.* 5, pp. 135-146. DOI: https://doi.org/10.1162/tacl_a_00051

[7] Devlin, J.; Chang, M.; Lee, K. and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers).* Minneapolis, Minnesota June 2019. Association for Computational Linguistics DOI: 10.18653/v1/N19-1423

[8] Pan, Y.; Xu, Q. and Li, Y. 2020. Food Recipe Alternation and Generation with Natural Language Processing Techniques. *Proceedings of the 2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW).* Dallas, Texas 20-24 April 2020. New York: IEEE. DOI 10.1109/ICDEW49219.2020.000-1

[9] Bishop, C. 2006. *Pattern recognition and machine learning*. Berlin: Springer. ISBN 0-387-31073-8.

[10] Schmidt, D. 2019. Simplified-recipes-1M Dataset. *dominik schmidt.* Available at: https://dominikschmidt.xyz/simplified-recipes-1M/ [Accessed: 15th January 2022]

[11] Gensim. 2022. *Topic Modelling for Humans.* Available at: https://radimrehurek.com/gensim/ [Accessed: 24th February 2022]

[12] Stack overflow. 2020. Loss does not decrease during training (Word2Vec, Gensim). Available at: https://stackoverflow.com/questions/52038651/loss-does-not-decrease-during-training-word2vec-gensim [Accessed: 27th April 2022]

[13] Van der Maaten, L. and Hinton, G. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research.* 9, pp. 2579-2605.

[14] Double check Vegan. 2021. *Vegan Ingredient Checker.* Available at: https://doublecheckvegan.com/ [Accessed: 5th March 2022]

[15] SciKit learn. 2022. *Scikit learn: Machine Learning in Python.* Available at: https://scikit-learn.org/stable/ [Accessed: 28th March 2022]

[16] Keras. 2022. *Keras.* Available at: https://keras.io/ [Accessed: 02 April 2022]

[17] Tensor Flow. 2022. *Classification on Imbalanced Data.* Available at: https://www.tensorflow.org/tutorials/structured_data/imbalanced_data [Accessed: 04 April 2022]

[18] Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I. and Salahutdinov, R. R. 2012. Improving neural networks by preventing co-adaption of feature detectors. *arXiv.* arXiv:1207.0580. DOI: https://doi.org/10.48550/arXiv.1207.0580

[19] Joachim, D. *The Food Substitutions Bible.* Second Edition. Ontario, Canada: Robert Rose.

[20] Watson-Price, E. 2022. *Alternatives to Pita Bread.* Available at: https://www.ehow.com/info_8185877_alternatives-pita-bread.html [Accessed: 17th May 2022]

[21] Dorsey, L. 2022. *Substitutes for Hazelnuts- what can I use instead?* Available at: https://www.supperforasteal.com/substitutes-for-hazelnuts/ [Accessed: 17th May 2022]

[22] The Low Carb Grocery. 2022. *Replacing Potatoes: Low carb alternatives.* Available at: https://www.thelowcarbgrocery.com/low-carb-lifestyle-blog/general-interest/7-sensational-low-carb-potato-substitutions [Accessed: 17th May 2022]

[23] Royce, W. W. 1970. Managing the Development of Large Software Systems. *Proceedings of IEEE WESCON 1970.* Los Angeles, 25-28 August. New York: IEEE.

[24] Python. 2022. *About Python*. Available at: https://www.python.org/about/ [Accessed: 20th May 2022]

[25] Salvador, A. *Et al.* 2019. Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* DOI: https://doi.org/10.48550/arXiv.1810.06553

# Supporting Figures

**Figure 1. T-SNE Dimensional Reduction of FastText Model, 50 Most Frequent Ingredients**

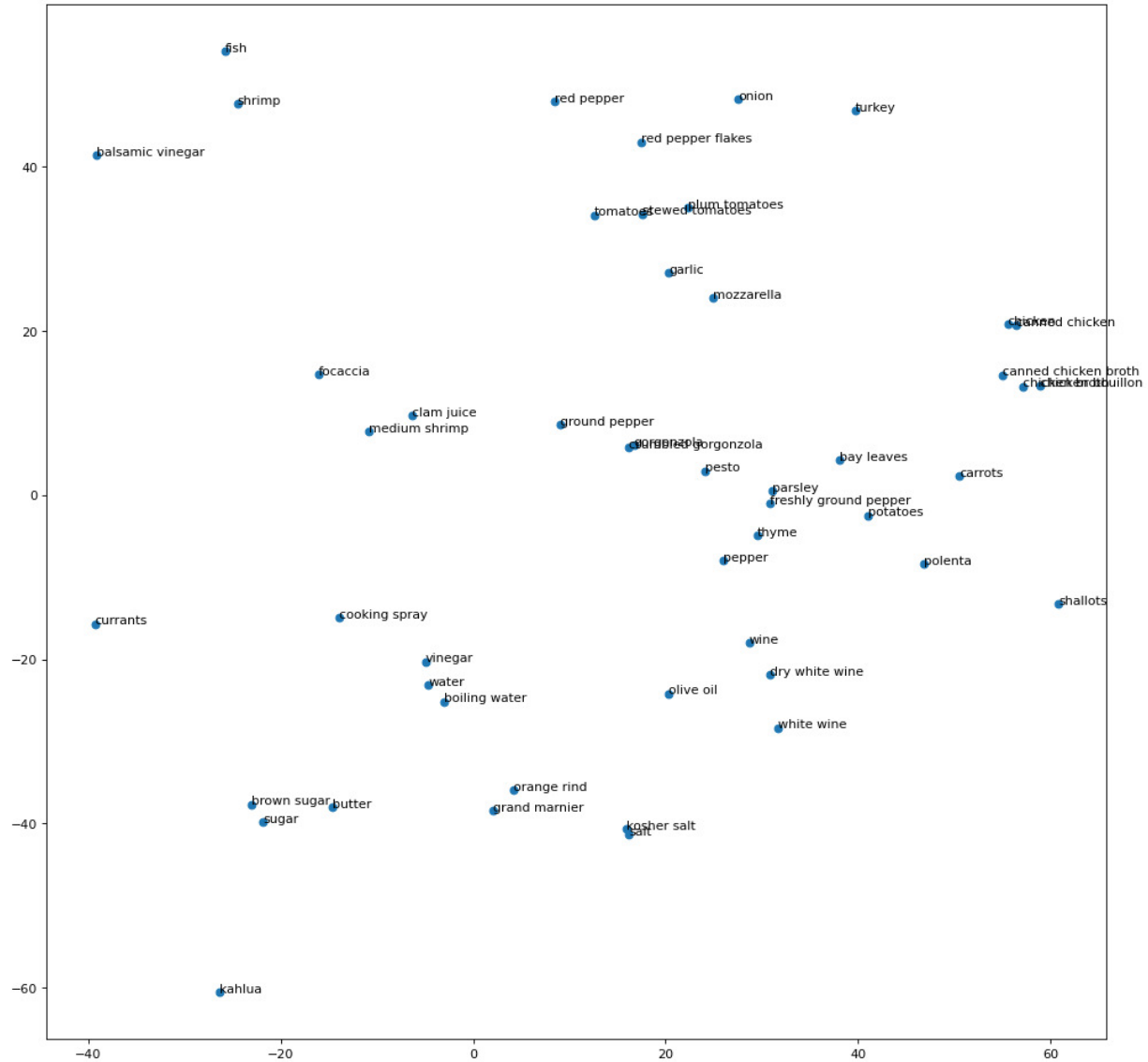



**Figure 2. Example Output of ingredient_filter Function.**

```
ingredient_filter(substituter = "fasttext",
```

```
classifier = "knn",
search_term = "sausage",
blacklisted_term = "",
n = 10,
filters = ["vegan", "vegetarian"],
verbose = 1)
```

| Index | Ingredient | Similarity | Classification |
|---|---|---|---|
| 0 | mild sausage | 0.874383 | carnivore |
| 1 | sausage links | 0.871964 | carnivore |
| 2 | spicy sausage | 0.849921 | carnivore |
| 3 | sausage meat | 0.836955 | carnivore |
| 4 | sausages | 0.826906 | carnivore |
| 5 | ground sausage | 0.807936 | carnivore |
| 6 | sausage casings | 0.804782 | carnivore |
| 7 | hot sausage | 0.794037 | carnivore |
| 8 | pork sausage | 0.790287 | carnivore |
| 9 | italian turkey sausage | 0.774680 | carnivore |

**Figure 3. Ground Truth Evaluation Table**

| Ingredient | Substitutions |
|---|---|
| Bacon Slices | Pancetta, Prosciutto, Salt Pork, Smoked Sausage, Smoked Ham, Black Forest Ham, Fatback |
| Cream | Yoghurt, Creme Fresh, Sour Cream, Milk, Evaporated Milk |
| Olive oil | Corn Oil, Sesame Seed Oil, Canola Oil, Butter Oil, Rapeseed Oil, Grapeseed Oil, Safflower Oil, Vegetable Oil |
| Corn Syrup | Golden Syrup, Maple Syrup, Sugar, Brown Sugar, Honey, Agave Syrup, Cane Syrup |
| Ricotta cheese | Cottage Cheese, Cream Cheese, Mascarpone Cheese, Sour Cream, Goat's Cheese, Greek Yoghurt, Mozzarella Cheese, Paneer, Parmesan Cheese |
| Brown lentils | Puy Lentils, Red Lentils, Green Lentils, Chickpeas, Black Beans, Pinto Beans, Lima Beans, Fava Beans |
| Chicken | Turkey, Rabbit, Quail |

| | |
|---|---|
| breast | |
| Green onion | Scallions, Leeks, Shallots, Chives, Red Onions, White Onions, Yellow Onions |
| Noodles | Linguine, Fettuccine, Pappardelle, Rice |
| Potato | Parsnips, Sweet Potato, Turnips, Celery Root, Radish |
| Smoked Salmon | Salmon, Smoked Trout |
| Hazelnuts | Almonds, Cashews, Walnuts |
| Lamb neck | Beef, Pork, Viel, Mutton |
| Pita bread | Lavash, Naan, Focaccia, Flour Tortilla, Pizza Crust |
| Coconut milk | Coconut Cream, Powdered Coconut Cream, Water, Milk, Cream |