“Feed Me”

Predicting potentially problematic ingredients from menu descriptions

By Sahil Sanghvi & Jacqueline Zhang

Purpose and Audience

Purpose

Dietary restrictions can make ordering food at some restaurants confusing, and perhaps even stressful.

By using a dish's name and description, which may (but won't always) contain hints to the ingredients, can we predict if that dish contains ingredients that may violate dietary restrictions?

Purpose

The FDA has six food allergen categories, and we also included meat for vegetarians and vegans.

- Peanuts
- Egg
- Sesame
- Fish
- Shellfish
- Soy
- Meat

Audience

People who have dietary restrictions:

- Vegetarians
- Vegans
- Pescatarians
- People who have food allergies
- Dogs (can’t have grapes)
- Etc.

Data

Table: `dishes.csv`

`dish_id`	Unique identifier
`menu_section`	String
`dish_name`	String
`menu_description`	String

Table: `dish_ingredients.csv`

`dish_id`	Unique identifier
`ingredient_id`	Unique identifier
`serving_size`	Float, units = grams

Table: `ingredients.csv`

`ingredient_id`	Unique identifier
`ingredient_name`	String

dishes.csv

5a241239-e6bd-415d-ae3d-59ab678e3992	Wraps	Mediterranean	Fresh lettuce, tomatoes, onions, olives, cucumber, feta cheese and Greek dressing.
c1ebf3e9-d6fa-4f3d-b0fa-eaba9d88163e	Burgers & Sandwiches	Bacon Cheeseburger	Served with lettuce & tomato.
12d90ec8-94f6-49c4-b26e-1e3c869f36aa	Drinks	Naked Juice	
910c030f-31d1-4021-9749-b4cc22f0d40b	Homemade Breads	Garlic Naan	Unleavened, handmade bread stuffed with fresh garlic.
a5fb5273-ff25-4475-a296-632a4bb15f1d	Chicken Hot Entrees	Chicken Parmesan	Served with penne pasta and marinara sauce
7e872e22-1420-4fbf-b792-0092eaf0effa	Symphony's Signature	Double Match Maki	Sweet potato tempura, fresh pineapple cover with baked salmon and shredded sweet potato on wasabi mayo sauce.

`dish_ingredients.csv`

4fe1311a-504d-4562-8e2d-77ce61ee7581	96b561b5-784f-431f-a2df-6ee4517f8fdd	0.9
4fe1311a-504d-4562-8e2d-77ce61ee7581	c67ee877-2b18-4cc7-beb2-7372da69a3ed	106.6
4fe1311a-504d-4562-8e2d-77ce61ee7581	becf2917-ea22-4786-8c61-99d08e11d631	2.4
4fe1311a-504d-4562-8e2d-77ce61ee7581	c93a73e1-dba3-4136-b690-2b3e991c460f	12.4
4fe1311a-504d-4562-8e2d-77ce61ee7581	e28c8608-ccd2-4392-baef-7759befe431f	24.8
4fe1311a-504d-4562-8e2d-77ce61ee7581	2afcadab-5250-411f-b19e-3243a1347a59	38.1

Each dish contains many items, which is why the first column (dish_id) has the same value for many rows.

ingredients.csv

96b561b5-784f-431f-a2df-6ee4517f8fdd	salt, table
d0df1e63-bbbc-4ca9-9f5e-099407813176	butter, without salt
6aa96588-e38a-45d8-a029-51d3099c11d0	olive oil
b6e9211b-2800-4986-9af8-35b65ec39711	sugars, granulated
6de730ce-7de1-4441-b4b7-4122e354650c	egg, whole, raw, fresh
1ff1becf-b7ca-4e89-b03e-52f1d7c4bccc	garlic, raw

80,032 dishes

7,295 ingredients

The data isn't perfect

While most of the data was pretty high-quality, there were a few records that seem to miss the mark a little bit.

E.g.: The dish_ingredients data says that the Chicken Bacon Ranch dish only has 1 ingredient associated with that, and that ingredient is salad dressing, ranch dressing, regular.

But for our purpose, it's pretty important to note that clearly there is meat somewhere in there.

Preprocessing

Preprocessing

» Remove rows with NaN / blank menu descriptions

Preprocessing

- Remove rows with NaN / blank menu descriptions
- Subsample to 20,000 random records.

Preprocessing

- Remove rows with NaN / blank menu descriptions
- Subsample to 20,000 random records.
- Remove stop words: “the”, “is”, “at”, ...

Preprocessing

- Remove rows with NaN / blank menu descriptions
- Subsample to 20,000 random records.
- Remove stop words: “the”, “is”, “at”, ...
- Stem remaining words: “prepared”, “preparation” -> “prepar”

Preprocessing

- Remove rows with NaN / blank menu descriptions
- Subsample to 20,000 random records.
- Remove stop words: “the”, “is”, “at”, ...
- Stem remaining words: “prepared”, “preparation” -> “prepar”
- Create 0/1 columns for each of our target ingredients.

Tokenizing Data

Approach 1:

Fit a **tokenizer** (`keras.preprocessing.text.Tokenizer`) to the descriptions, and use that to generate encoding vectors for each description.

Approach 2:

Use a measure called **Term Frequency, Inverse Document Frequency (tf-idf)** (`sklearn.feature_extraction.text.TfidfVectorizer`) with a `min_df` value of 5 and a `ngram_range` of (1, 2), and use that to generate encoding vectors for each description.

APPROACH #1

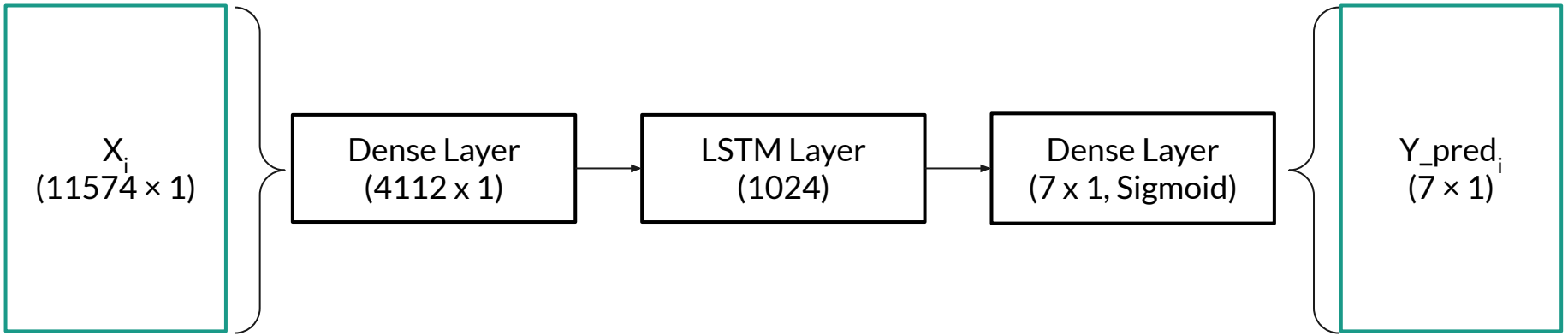
Neural Networks

RNN

Train a recurrent neural network to predict a 7-dimensional column vector of 0's and 1's. (One for each ingredient).

To get the input from a string of many words to a numerical vector, we used Term Frequency, Inverse Document Frequency (tf-idf) as explained earlier.

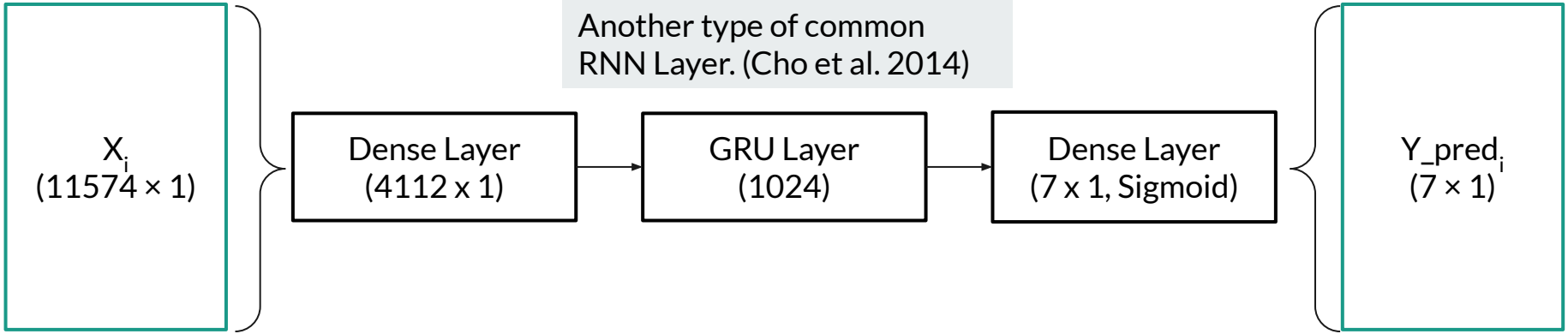
RNN #1



Across all hyperparameter tuning, validation accuracy hovered very close to 0.53.

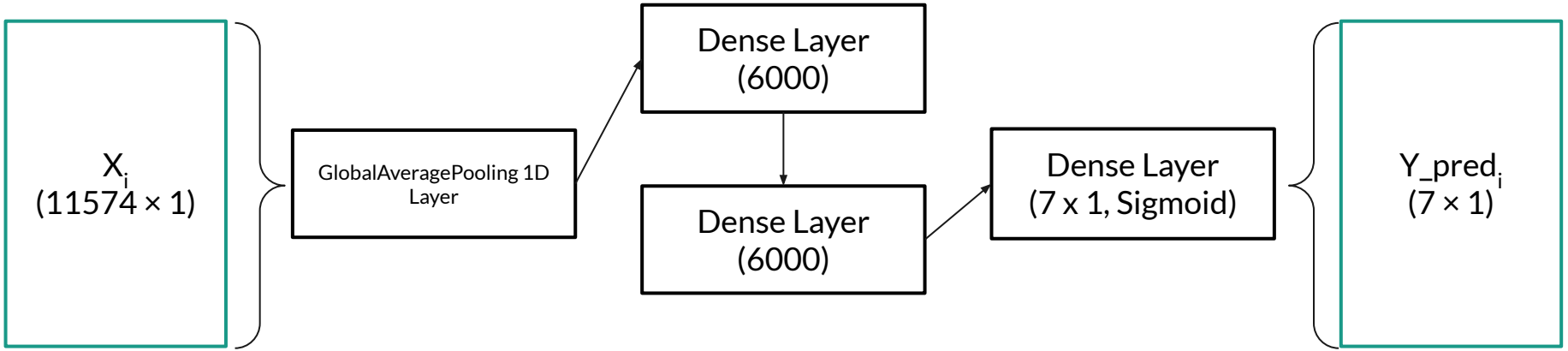
RNN #2

GRU = Gated Recurrent Unit. Another type of common RNN Layer. (Cho et al. 2014)



Across all hyperparameter tuning, validation accuracy hovered very close to ~0.57.

Sequential Pooling NN



Across all hyperparameter tuning, validation accuracy hovered very close to 0.54.

So long, Neural Nets!

Neural networks aren’t working too well for this. Let’s see what else we might be able to use!

APPROACH #2

Random Forest

Random Forest

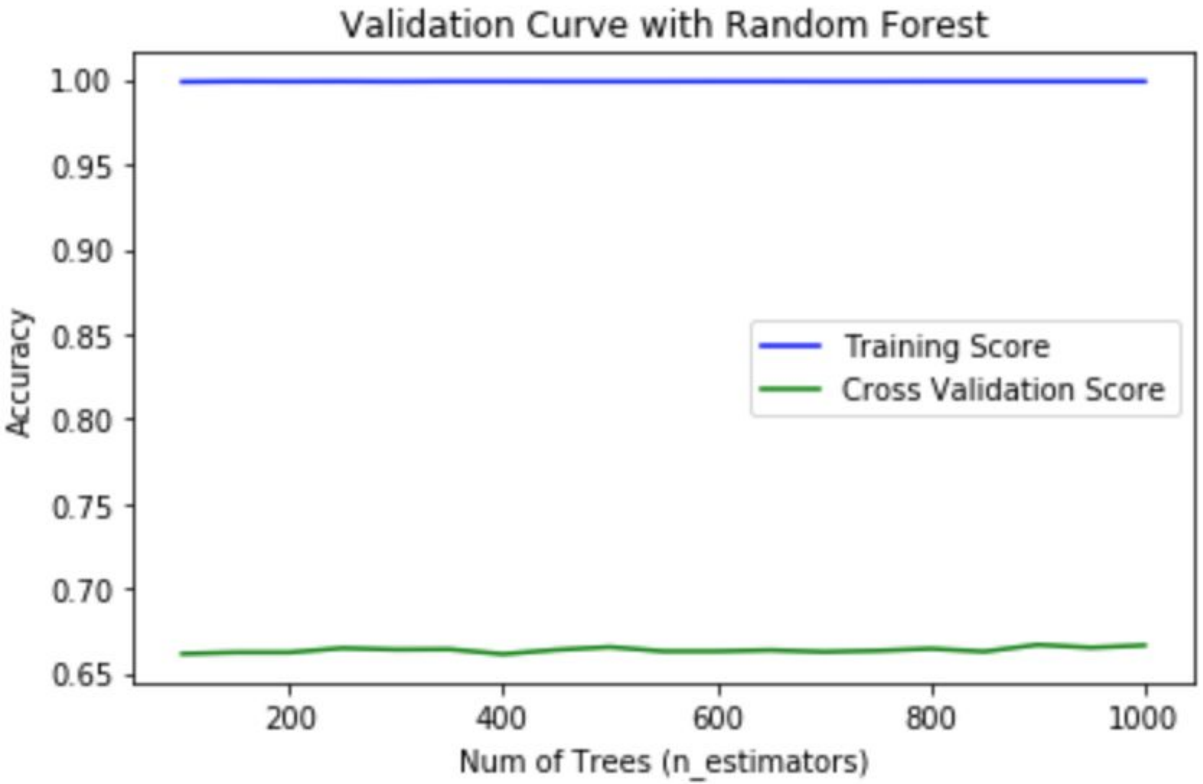
Train a random forest to predict a 7-dimensional column vector of 0's and 1's. (One for each ingredient).

To get the input from a string of many words to a numerical vector, we used Term Frequency, Inverse Document Frequency (tf-idf) as explained earlier.

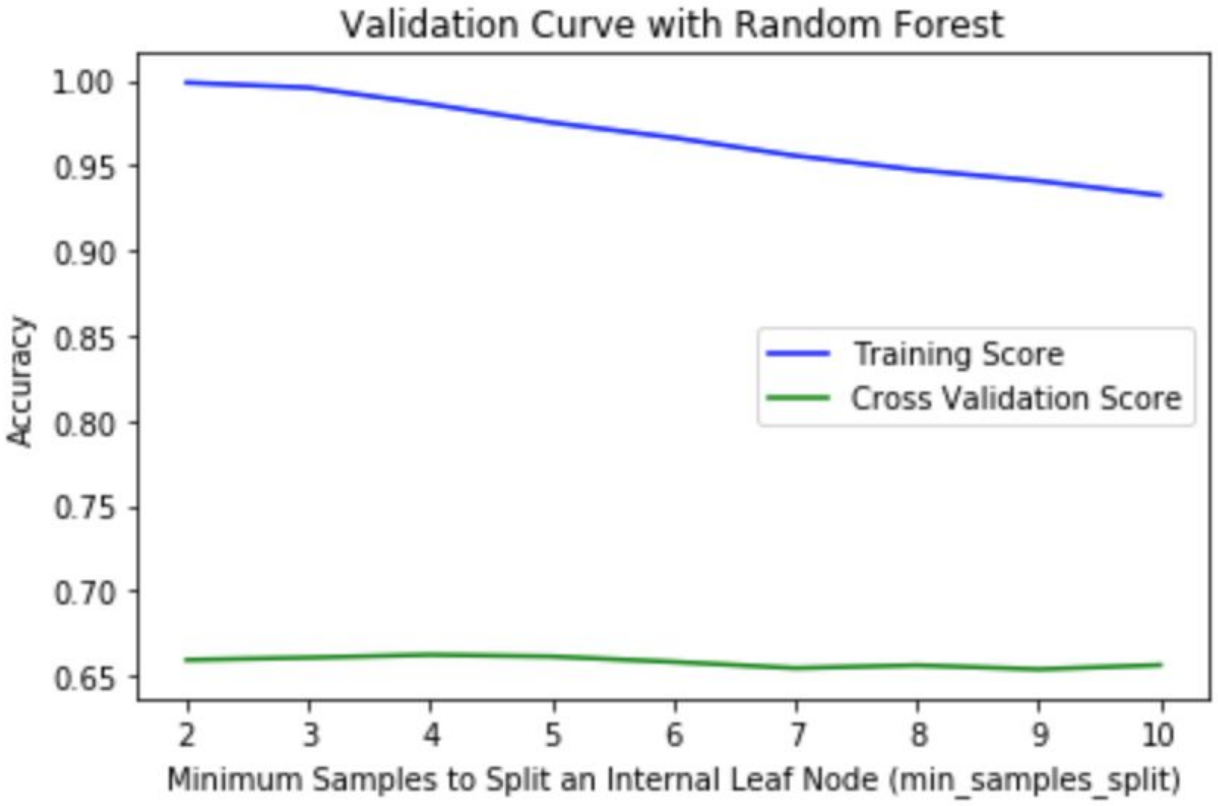
Hyperparameter Tuning

We chose to focus on 4 hyperparameters:

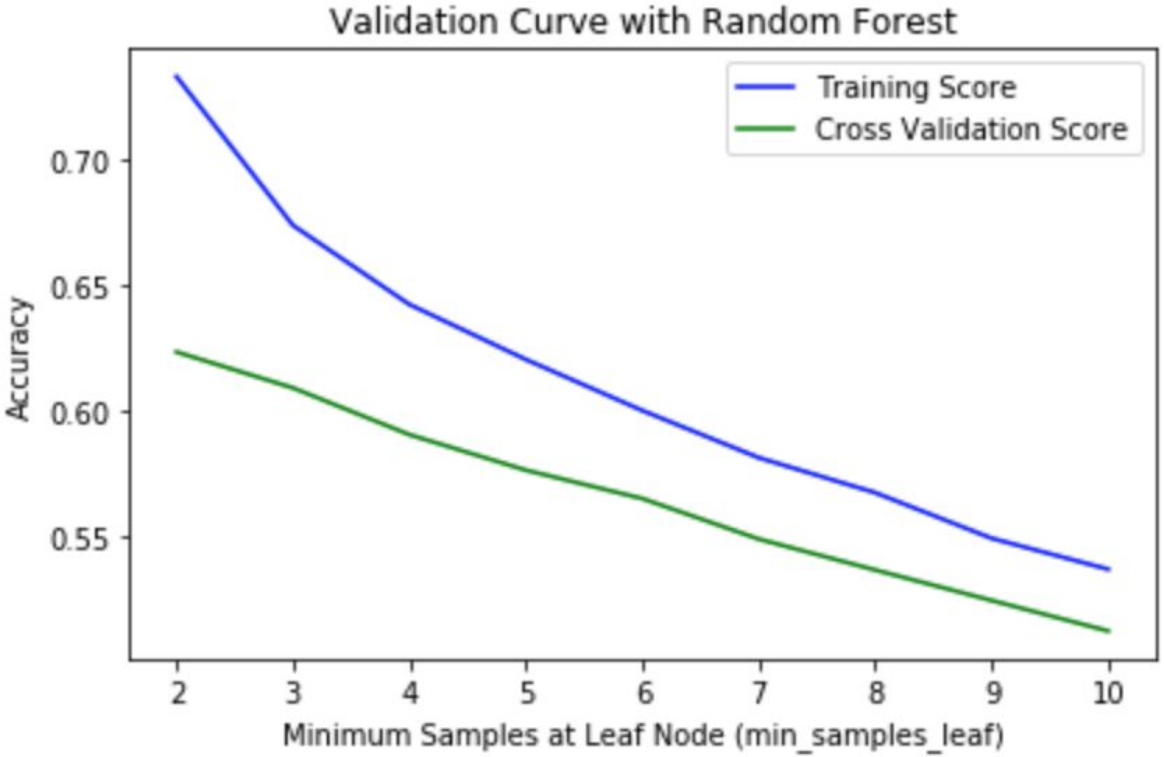
- `n_estimators` : specifies the number of trees in the forest of the model
- `min_samples_split` : the minimum number of samples required to split an internal leaf node
- `min_samples_leaf` : the minimum number of samples required to be at a leaf node
- `max_depth` : maximum depth of each tree



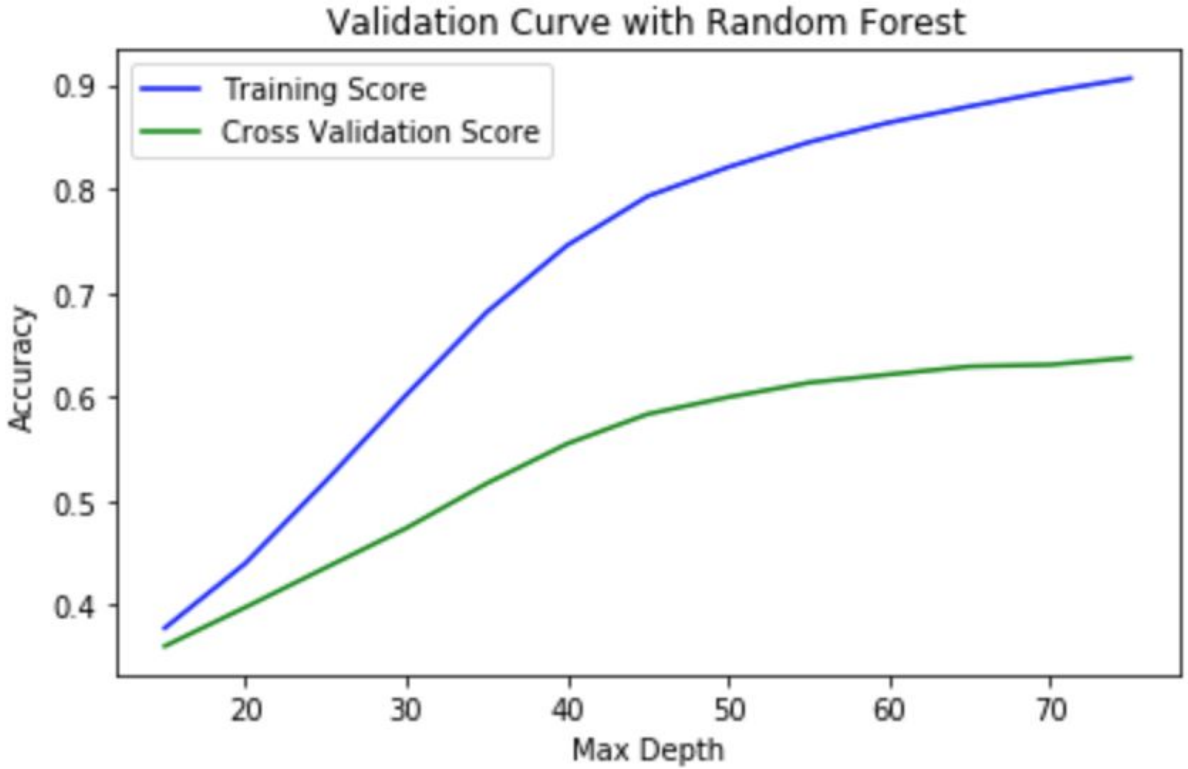
It is clear that changing the number of estimators did not make a huge difference to the validation score. Regardless, we chose the `n_estimators` value with the highest cross validation accuracy, which was 900.



It is interesting how the training score decreases as the `min_samples_split` value increases, while the cross validation score remains pretty steady in the ~0.65 accuracy. We chose the `min_samples_split` value with the highest cross validation accuracy, which was 4.



It is clear that as the number of `min_samples_leaf` increases, the training and the cross validation score decreases. We chose the `min_samples_leaf` value with the highest cross validation accuracy, which was 2.



Finally, we tuned the `max_depth` value. The values start plateauing around 55 for the cross validation score, so we chose to use 55 as our `max_depth` value.

only... 59.25%

So long, Random Forests!

Random forests aren't working too well for this. Let's see what else we might be able to use!

APPROACH #3

Multi-Label Classifiers

3 Models → 2 Learning Algorithms

Logistic Regression

Naive Bayes

Support Vector Machine (SVM)



Binary Relevance
creates an ensemble of single-label binary classifiers, one for each class, that are trained

One vs Rest
decomposes data into multiple independent binary classification problems

Why do we need to pair these models with those learning algorithms?

Binary Relevance Results (Accuracy)

Logistic Regression	Naive Bayes	SVM
0.625	0.5605	0.701

One vs Rest Results (Accuracy)

Model	Contains Peanuts	Contains Egg	Contains Sesame	Contains Fish	Contains Shellfish	Contains Soy	Contains Meat
Logistic Regression	0.9635	0.8585	0.93	0.9765	0.961	0.9115	0.8765
Naive Bayes	0.9615	0.851	0.929	0.9635	0.9455	0.912	0.831
SVM	0.977	0.8935	0.9495	0.989	0.985	0.9365	0.8865

APPROACH #4

Neural Networks...again

Ok, hear me out.

So having a neural network to predict all 7 indicators at once didn’t work well. But what if we try to only predict one at a time?

In [14]:
```
peanut_model.fit(
    peanut_X_train,
    peanut_Y_train,
    epochs = 5,
    batch_size=128,
    shuffle=True
)
```

```
Train on 15000 samples
Epoch 1/5
15000/15000 [==============================] - 31s 2ms/sample - loss: 0.1168 - accuracy: 0.9563 - false_positives: 91.0000 - fa
lse_negatives: 565.0000
Epoch 2/5
15000/15000 [==============================] - 31s 2ms/sample - loss: 0.0349 - accuracy: 0.9853 - false_positives: 33.0000 - fa
lse_negatives: 188.0000
Epoch 3/5
15000/15000 [==============================] - 32s 2ms/sample - loss: 0.0101 - accuracy: 0.9969 - false_positives: 24.0000 - fa
lse_negatives: 22.0000
Epoch 4/5
15000/15000 [==============================] - 34s 2ms/sample - loss: 0.0053 - accuracy: 0.9987 - false_positives: 8.0000 - fal
se_negatives: 11.0000
Epoch 5/5
15000/15000 [==============================] - 33s 2ms/sample - loss: 0.0025 - accuracy: 0.9991 - false_positives: 5.0000 - fal
se_negatives: 8.0000
```

Out[14]:
```
<tensorflow.python.keras.callbacks.History at 0x20a410a92c8>
```

In [15]:
```
peanut_model.evaluate(
    peanut_X_test,
    peanut_Y_test
)
```

```
5000/5000 [==============================] - 6s 1ms/sample - loss: 0.1727 - accuracy: 0.9772 - false_positives: 35.0000 - false
_negatives: 79.0000
```

In [14]:
```
peanut_model.fit(
    peanut_X_train,
    peanut_Y_train,
    epochs = 5,
    batch_size=128,
    shuffle=True
)
```




```
Train on 15000 samples
Epoch 1/5
15000/15000 [==========
lse_negatives: 565.0000
Epoch 2/5
15000/15000 [==========
lse_negatives: 188.0000
Epoch 3/5
15000/15000 [==========
lse_negatives: 22.0000
Epoch 4/5
15000/15000 [==========
se_negatives: 11.0000
Epoch 5/5
15000/15000 [==========
se_negatives: 8.0000
```

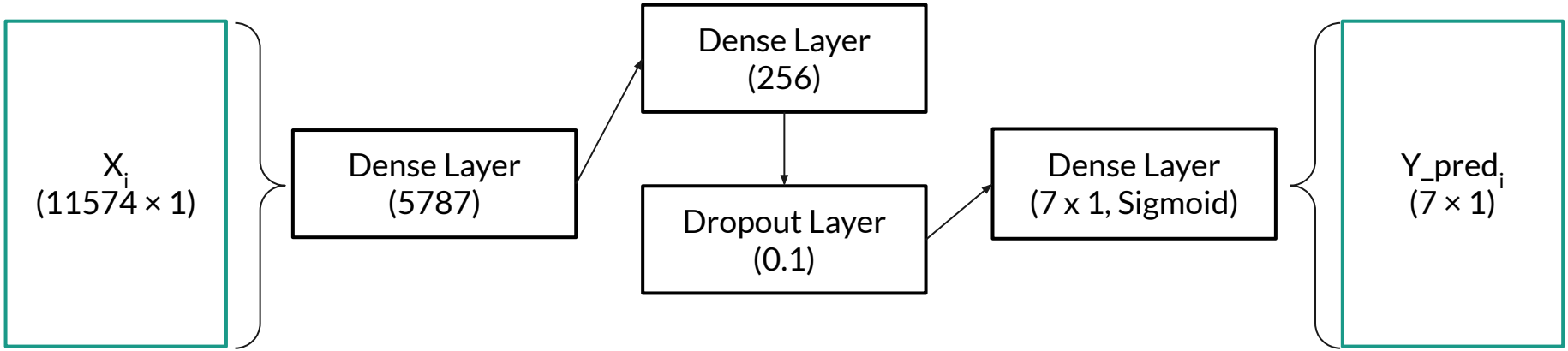
```
false_positives: 91.0000 - fa
false_positives: 33.0000 - fa
false_positives: 24.0000 - fa
false_positives: 8.0000 - fal
false_positives: 5.0000 - fal
```

Out[14]: <tensorflow.python.keras

In [15]:
```
peanut_model.evaluate(
    peanut_X_test,
    peanut_Y_test
)
```

```
5000/5000 [==============================] - 6s 1ms/sample - loss: 0.1727 - accuracy: 0.9772 - false_positives: 35.0000 - false
_negatives: 79.0000
```

The correct answer is often the most simple...



This architecture worked well for peanuts and fish. Will it perhaps work well for all of them?

	Ingredient	Validation BinaryCrossEntropy Loss	Validation Accuracy	Validation False Positives	Validation False Negatives
0	peanuts	0.154269	0.9760	39.0	81.0
1	egg	0.085323	0.9852	37.0	37.0
2	sesame	0.289691	0.9500	100.0	150.0
3	fish	0.077680	0.9862	22.0	47.0
4	shellfish	0.103821	0.9744	62.0	66.0
5	soy	0.369993	0.9258	172.0	199.0
6	meat	0.481281	0.8896	243.0	309.0

	Ingredient	Validation BinaryCrossEntropy Loss	Validation Accuracy	Validation False Positives	Validation False Negatives
0	peanuts	0.154269	0.9760	39.0	81.0
1	egg	0.085323	0.9852	37.0	37.0
2	sesame	0.289691	0.9500	100.0	150.0
3	fish	0.077680	0.9862	22.0	47.0
4	shellfish	0.103821	0.9744	62.0	66.0
5	soy	0.369993	0.9258	172.0	199.0
6	meat	0.481281	0.8896	243.0	309.0

Good results for four of the ingredients, but sesame, soy, and meat still need improvement.

APPROACH #5

Ensembling

What if we created individual classifiers for these models and combined them?

Neural Network

Random Forest

SVM

	Ingredient	Validation Accuracy	Validation False Positives	Validation False Negatives
0	peanuts	0.9794	18	85
1	egg	0.9888	20	36
2	sesame	0.9556	↓ 82	↓ 140
3	fish	0.9884	23	35
4	shellfish	0.9812	27	67
5	soy	0.9336	↓ 148	↓ 184
6	meat	0.8950	↑ 261	↓ 264

	Ingredient	Validation Accuracy	Validation False Positives	Validation False Negatives
0	peanuts	0.9794	18	85
1	egg	0.9888	20	36
2	sesame	0.9556	82	140
3	fish	0.9884	23	35
4	shellfish	0.9812	27	67
5	soy	0.9336	148	184
6	meat	0.8950	261	264

Ready-to-ship?

Not exactly. This project is meant for quick, simple, everyday use, which is not one of Jupyter Notebook's strong suits.

We'd like to package this up into a smartphone app, and distribute it that way for on-the-go use.

A potential avenue for improvement, if we make a smartphone app, is to use the phone camera to take a picture of the menu and run text recognition on it, automatically capturing the needed data.