Machine Learning

Predicting the presense of starch in food

Peter Tran

Table of Contents

Data Pre-processing

Before applying machine learning models, it is important for the dataset to be properly pre-processed.

Removing columns with low data count

I have decided to only look at the ‘All solids & liquids per 100g’ sheet of the Excel file for my analysis. For the excel read was read and convert into a DataFrame object.

```
df = pd.read_excel('Rel_2_Nutrient_file.xlsx', sheet_name="All solids & liquids per 100g")
```

Upon first glance of the data, there are 1616 rows and 293 columns of data. There appears to be significant amount of missing data which shows as NaN. Interestingly, strings are not considered as NaN. Through running a review, it was found that 60.52% of total cells were missing data. However, it was discovered that the certain columns were more significantly dense than others, so an algorithm was run to remove columns that had less than 10% of the data point filled.

This removed 230 columns, however greatly reduced the percentage of missing data from 60.52% to 0.21% of total cells. Finally, there were 2 rows with missing data so they dropped from the DataFrame which now meant we had a valid and completely filled dataset with no NaN.

Correlation Matrix

A correlation matrix was calculated on the dataset.

```
corr_matrix = df.corr().abs()

sol = (corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
      .stack()
      .sort_values(ascending=False))

for index, value in sol.items():
   print(index, value)
```

There were many columns that had very highly correlation scores which suggests those columns may be direct functions of one to another. The top 13 correlations are recorded below in the table.

Columns 1	Columns 2	Correlation
Niacin derived from tryptophan (mg)	Tryptophan (mg)	0.9999958850241869
Available carbohydrate, without sugar alcohols (g)	Available carbohydrate, with sugar alcohols (g)	0.9993175077647121
Protein (g)	Nitrogen (g)	0.9983917080579181
Energy with dietary fibre, equated (kJ)	Energy, without dietary fibre, equated (kJ)	0.9976427909974308
Beta-carotene (ug)	Beta-carotene equivalents (provitamin A) (ug)	0.9903888312687489
Added sugars (g)	Free sugars (g)	0.9830406298958576
Retinol (preformed vitamin A) (ug)	Vitamin A retinol equivalents (ug)	0.9770433489721876
C22:6w3 (mg)	Total long chain omega 3 fatty acids, equated (mg)	0.9727333553422074
Total folates (ug)	Dietary folate equivalents (ug)	0.969123559575261
C20:5w3 (mg)	Total long chain omega 3 fatty acids, equated (mg)	0.9536367280404122
Niacin (B3) (mg)	Niacin derived equivalents (mg)	0.9519386346698158
C18:2w6 (g)	Total polyunsaturated fatty acids, equated (g)	0.9518442757394865
Sucrose(g)	Added sugars (g)	0.9473649910055121

As can be seen, columns such as “Niacin derived from tryptophan (mg)” and “Tryptophan (mg)” high an extremely high correlation and as such, indicate that one column is derived from the other.

Columns that have high correlations with other columns can make it hard for certain models such as linear regression and logistic regression to model. As such, one column from each pair of features that have a higher correlation that 0.8 were removed. 0.8 was picked as the threshold as this is defined at a correlation degree of ‘Very Strong’ iin the Grading table of Spearman as can be seen in the appendix. This removed 21 columns and now the dataset contains 42 columns totals.

Low Variance

It was checked for columns that have completely non-unique values and the column '25-hydroxy ergocalciferol (25-OH D2) \n(ug)' was identified as such. This column only had the value 0 and would add no information to the model so it was deleted from the data frame.

Unique Identifiers

Unique identifiers such as the features ‘Public Food Key’, ‘Classification’ and ‘Food Name’ should be removed when developing machine learning models as they are typically arbitrary and do not represent any underlying patterns in the data. Additionally, this can lead to overfitting in the model and would hinder the model’s ability to generalise from the training set.

Explanation of processed dataset

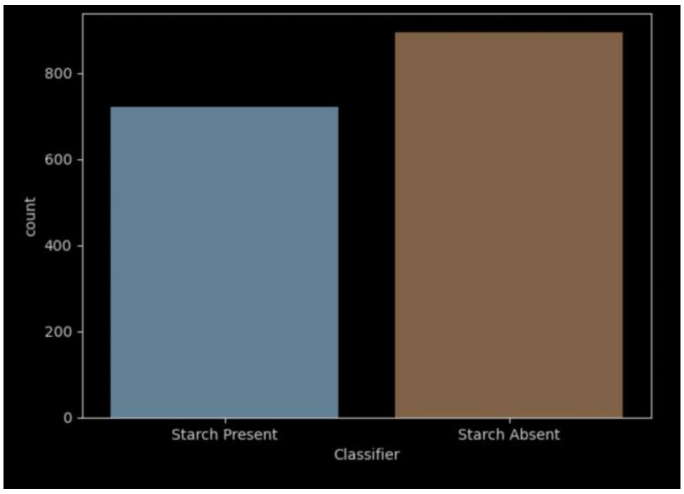
As a result of the steps above, the cleaned data set:

1. Contains features that had at least 90% of their rows populated
2. Does not contain any features that have a ‘Very Strong’ correlation with another feature
3. Does not contain any features that do not have any more than one unique value
4. Does not contain any features that are unique identifiers for the row
5. Contains 38 features and,
6. Contains 1614 rows

Problem Identification

This report will explore the task of creating a classification model for predicting the presence of Starch in different foods. Starch is an important macro and is an essential part of a nutritious diet so a model like this would be useful and relevant.

As starch is recorded as a continuous value as the weight in starch in the dataset, the feature first needs to be converted into distinct values so it can be used with a classification algorithm. As such, the classifications ‘Starch Present’ and ‘Starch Absent’ were used where ‘Starch Present’ represents more than 0g of Starch per 100g and ‘Starch Absent’ indicates no starch detected at all. Below shows the distribution of the classifications in the dataset and it is a good split of around 43% ‘Starch Present’ and 57 ‘Starch Absent’.



KNN

KNN is a non parametric instance based algorithm and has been chosen as one of the machine learning models to use for this classification problem as it easy to interpret and is good for datasets where the features are all scaled proportionally.

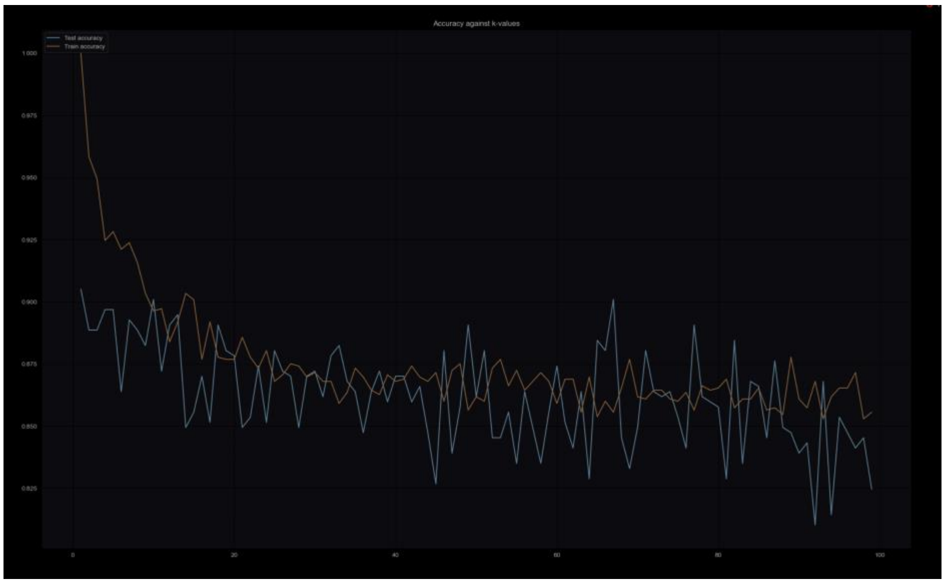
Naïve Trial

Running an initial trial of KNN model for this classification problem is important to develop a baseline and will provide a sense of how well the target classification, in this case, the presence of Starch, can be predicted using the features. From this, more sophisticated techniques can be used to tune the model and will provide a benchmark for subsequent models. As such, this first trial will use the default hyper parameters that is provided by sklearn which are N_neighours of 5 and using the minkowski metric. The test set size was set to 0.3.

Additionally, the model was ran 10 times to add variability in the data split and provide are more robust estimate of the model performance and stability. The result of this trial was a accuracy score of 0.862 for the test set and 0.928 for the train set over a trial of 10 runs.

Tuning Hyperparameter K

To improve the model, different values of the hyper parameter K was experimented to find at which value would result in the highest accuracy score. The range of K values tested were from 1 to 100 and the results can be seen below:



The highest accuracy score for this trial was found at K = 1, where for the test set, the accuracy score was 0.905 and for the train set, score of 1. Obviously, the accuracy for the train set would be perfect with a K value of 1 because each sample was classified by the nearest neighbour which is itself. As the K value increases in value, the model seems to fluctuate around the 0.87 accuracy for the train set and slighter lower around 0.86 for the test set.

Although K value of 1 gives the best value, this may not be the best choice because of model stability. As only the nearest neighbour is considered, the model may be sensitive to small fluctuations in the training set and can lead to unstable prediction. As such, usually a larger K value is used for more robust models but this depends on the dataset.

Cross Validation

Cross validation is another method to evaluate a model's performance by partitioning the data into subsets and trains on all but one of the subsets at a time. This results in a better estimate of how the model performs on unseen data compared to just splitting the data on a fixed number. The results are below:

```
Cross-validation scores: [0.85185185 0.78395062 0.82098765 0.99382716 0.79503106 0.67701863]
Average cross-validation score: 0.795372287401273
```

As can be seen, the scores for the cross-validation fluctuations significantly between the trials and suggests that the performance may be highly dependent on what data is used for the training and can usually a sign of overfitting.

Grid Search Tuning

Grid Search is tuning method that can be used on a KNN model to brute force find the most effective parameters that will result in the highest accuracy score. The methods takes in combinations of the possible parameters and returns the parameter combination that has the highest cross validated accuracy.

The parameters that were given to GridSearch were:

```
grid_params = {'n_neighbors' : [x for x in range(1,100)],
           'algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute'],
           'weights' : ['distance', 'uniform'],
           'metric' : ['minkowski', 'euclidean', 'manhattan']}
```

These were the results of GridSearch with a fold of 10 for cross validation:

```
Fitting 5 folds for each of 2376 candidates, totalling 11880 fits
Best Score:  0.9349655850540807
Best Parameters:  {'algorithm': 'auto', 'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}
```

Below shows the comparison between the best parameters and default parameters of sklearn KNeighborsClassifier()

	Benchmark	**Tuned**
Algorithm	Auto	Auto
Metric	Minkowski	Manhattan
N_neighbors (K)	5	3
Weights	Uniform	Distance

This score is 7.29% better than the benchmark KNN model using the default parameters. Overall, 0.935 can be considered high however, the accuracy score may be improved further if another machine learning is considered. KNN is a relatively simple model and there is not much more than can be done to significantly tune for a sufficiently greater accuracy score.

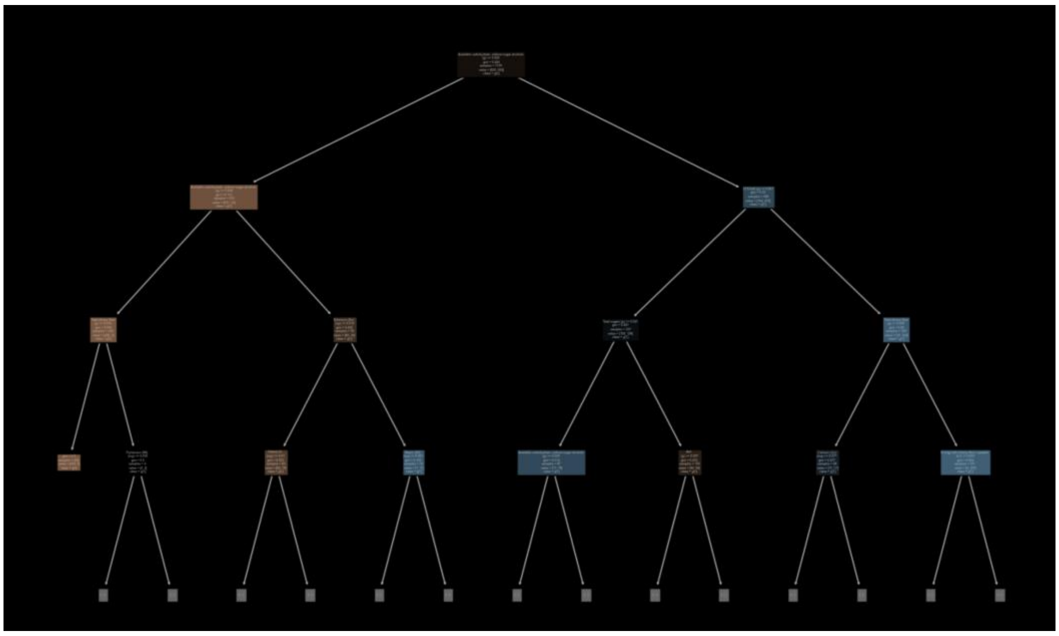
Decision Trees

Decision trees are another intuitive machine learning algorithm that can be used for classification problems. Decision trees may potentially be a better choice than KNN for the classification problem on this dataset as decision trees can model complex nonlinear relationships better than KNN.

Naïve Trial

Similarly, with KNN, a benchmark model is established first. Running the decision tree classifier from sklearn with the default parameters gives an accuracy score over 10 trials gives an average accuracy score of 0.891.

Below is a plot of a decision tree from the trials. Note that due to the magnitude of amount of features, the texts are small and difficult to read, however it gives good insight into which features are most important. The algorithm splits features based on the largest improvement to the prediction accuracy so features closer to the root are the most important. An insight that can be seen in this case is that the 'Available Carbohydrates, without sugar alcohol' has the most effect on the decision, which makes sense as starch is a carbohydrate so it is intuitive that there may exists a relationship between these features.



Grid Search Tuning

The hyper parameters of the Decision Tree classifier can be altered to attempt to improve the accuracy score. One such technique is called Pruning and involves reducing the size of the decision tree and to reduce overfitting, ultimately better generalisation of unseen data.

A grid search on the Decision Tree classifier was trialled with a cross fold value of 10 and the possible parameters of:

```
param_grid = {'max_depth': np.arange(3, 10),
              'min_samples_split': np.arange(2, 15),
              'min_samples_leaf': np.arange(1, 15),
              'criterion': ['gini', 'entropy']}
```

Best Score: 0.91232132
Best Parameters: {'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}

Below shows the comparison between the best parameters and default parameters of sklearn KNeighborsClassifier()

	Benchmark	**Tuned**
Criterion	Gini	Gini
Max Depth	None	3
Min Sample Split	2	2
Min sample Leaf	1	1

Overall, the accuracy scores only increased slightly from the benchmark average score of 0.891 to 0.912, which is a difference of 0.021. This is an insignificant change and means that the parameters of the naïve attempt was already close to optimised and this is supported by the table above which shows that only the Max Depth parameter was adjusted.

This accuracy score of 0.912 for this decision tree classifier results pale in comparison to the KNN model which overall, produced an accuracy score of approximately 0.935 which may be attributed to the fact that decision trees are prone to overfitting and possibly because the dataset has significant amount of noise in the data. KNN is usually more resistant to noise, especially at higher k values because the model classifies based on multiple neighbours and inherently averages out the noise.

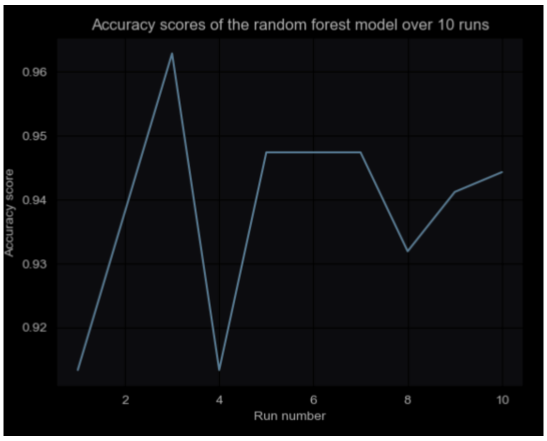
Random Forests

Although decision trees are powerful for certain situation, they can lead to overfitting. Random Forests can alleviate this issue due to using techniques such as bagging which creates multiple subsets of the original data and then trains a separate decision tree on each subset. As a result, this leads to reduced variance and overfitting.

Additionally, random forests introduce feature randomness which makes the model more robust and less likely to overfit.

Naïve Trail

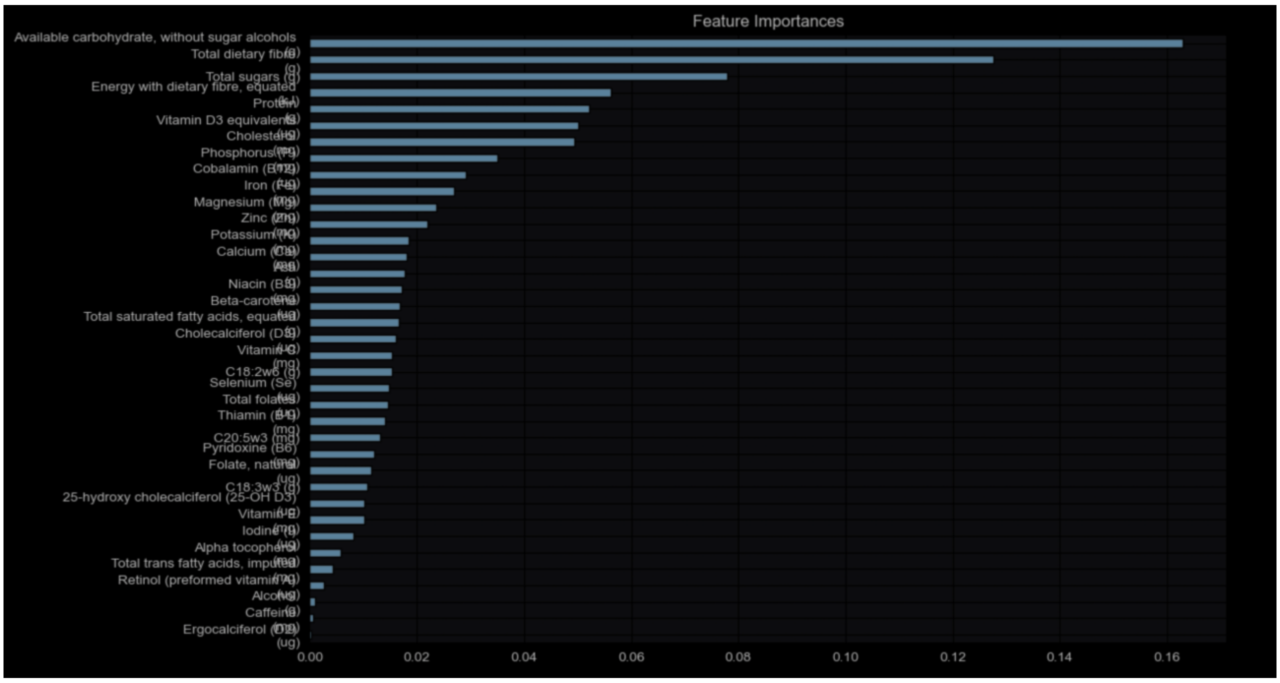
For this naïve attempt, the default hyperparameters were not changes and was trialled over 10 runs to find the average. Overall, the average accuracy score was 0.938 for the random forest classifier. Below is a plot showing the average accuracy scores over the 10 runs.



This is approximately the same accuracy score as the tuned KNN classification model.

Feature Removal

Feature removal is a technique that involves removing features that are considered unimportant or irrelevant. Below shows the features based on their importance to improve the accuracy score of the model.



The lowest 10 important features were:

['Ergocalciferol (D2) \n(ug)', 'Alcohol \n(g)', 'Caffeine \n(mg)', '25-hydroxy cholecalciferol (25-OH D3) \n(ug)', 'Cholecalciferol (D3) \n(ug)', 'Vitamin D3 equivalents \n(ug)', 'Retinol (preformed vitamin A) \n(ug)' 'C20:5w3 (mg)', 'Cobalamin (B12) \n(ug)' 'Cholesterol \n(mg)']

These features were dropped from the dataset and the random forest classifier was trained again over 10 trials. **With the 10 features dropped, the average accurate score now increased to 0.955.**

Grid Search Tuning

Grid Search was conducted for the Random Forests to find the best hyperparameters to maximise the accuracy score. These are the possible hyperparameters that were inputted into the GridSearch function:

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

Best Score: 0.9673434
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}

Below shows the comparison between the best parameters and default parameters of sklearn RandomForestClassifier()

	Benchmark	**Tuned**
N estimators	100	300
Max depth	None	10
Min Sample Split	2	2
Min sample Leaf	1	1

As can be seen in the table, the hyperparameters of N estimator was tuned to 300 from the default value of 100 and the max depth was set to 10 from the default value of None.

Conclusion

Three different machine learning models were iteratively used predict the presence of Starch in food. Data processing was conducted first to prepare the data for modelling as there a significant portion of the cells were missing data and not all columns provided useful information. These included columns that had the same values for all rows, unique identifiers, and columns that had ‘Very Strong’ with another column (as per the Standard Grading table of Spearman Correlation Coefficient Correlations).

Thee three different models used were KNN, Decision Trees and Random Forests.

KNN model was the first and had decent performance with a high accuracy score of 0.89 for the simple trial with no adjustments to the hyperparameters. After performing GridSearch and tuning the hyperparameters, the accuracy score increased to 0.935 for KNN.

Decision Trees was used next as it was believed it was better choice than KNN for the classification problem on this dataset as decision trees can model complex nonlinear relationships more efficiently than KNN. However, even after tuning the hyper parameters, the accuracy score of 0.912 for the decision trees was lower than the score of KNN.

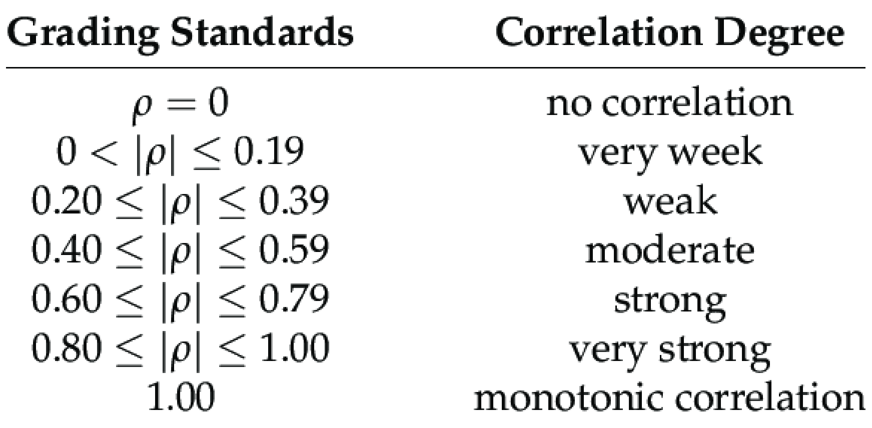
Random Forests was used next as evidence suggested that the decision tree was potentially overfitting and random forests had techniques to minimise this. After tuning the hyperparameters with GridSearch and conduct feature removal on features of low importance, the overall accuracy score of the random forest classifer was 0.967. This is the highest accuracy score discovered so far.

Final Accuracy Score and Hyperparameters

	Accuracy Score	Hyperparameters
KNN	0.935	'algorithm': 'auto', 'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'
Decision Trees	0.912	'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2
Random Forests	0.967	'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300

Appendix

Standard Grading table of Spearman correlation coefficient



Grading Standards	Correlation Degree
$\rho = 0$	no correlation
$0 < \lvert\rho\rvert \leq 0.19$	very week
$0.20 \leq \lvert\rho\rvert \leq 0.39$	weak
$0.40 \leq \lvert\rho\rvert \leq 0.59$	moderate
$0.60 \leq \lvert\rho\rvert \leq 0.79$	strong
$0.80 \leq \lvert\rho\rvert \leq 1.00$	very strong
1.00	monotonic correlation

Spearman's Rank-Order Correlation - A guide to when to use it, what it does and what the assumptions are. (2023). Available at: https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide.php (Accessed: 26 May 2023).

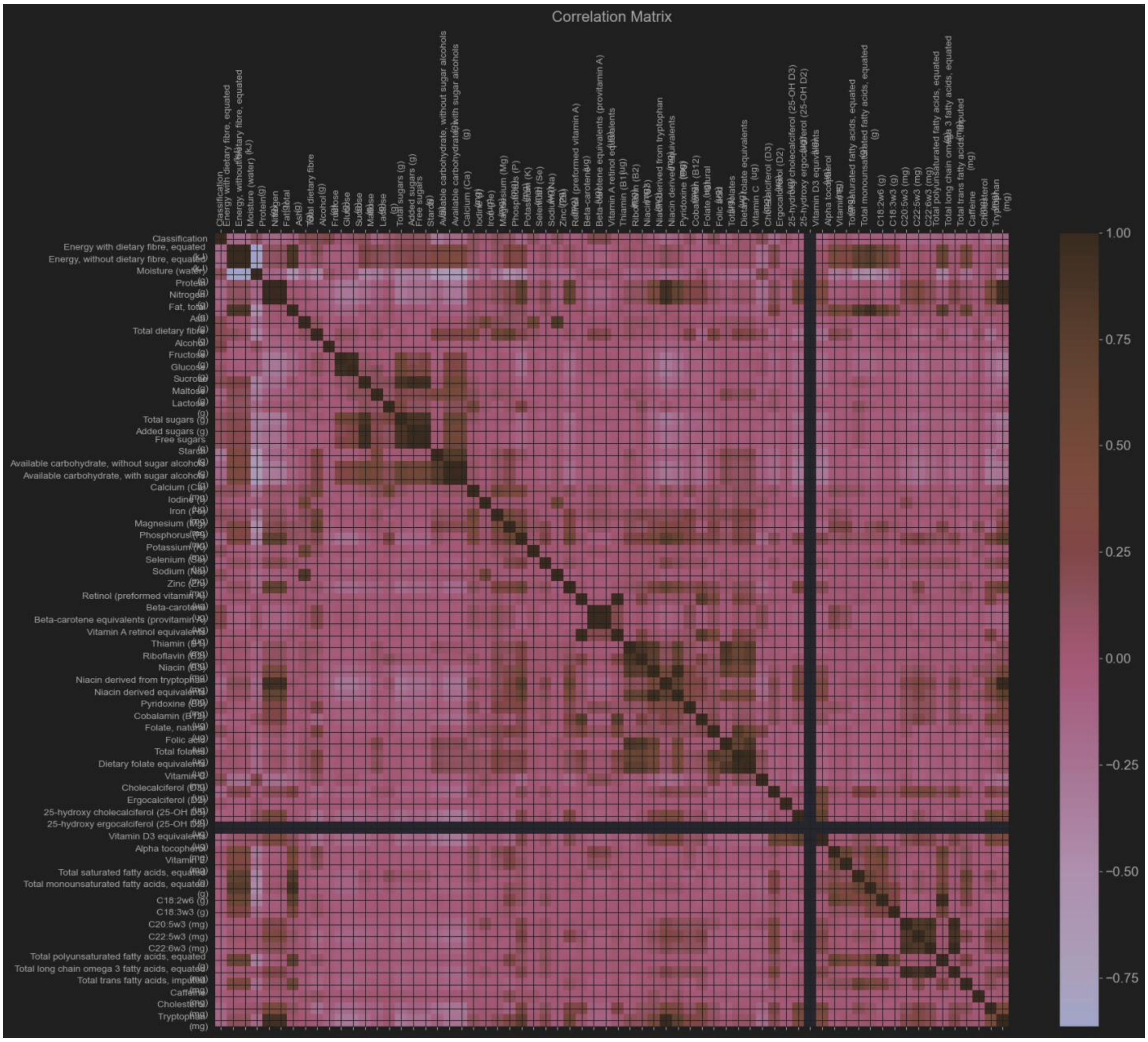
Data Pre-processing

Code to remove columns that have percentage of data less than 10%.

```
threshold = 0.1
cols_to_drop = df.columns[(df.count() / len(df)) < threshold]
df = df.drop(cols_to_drop, axis =1)

df = df[df.columns[df.isnull().mean() < 0.1]]
```

A correlation matrix heatmap was also generated however it is difficult to view properly due to the large number of columns.

```
f = plt.figure(figsize=(19, 15))
plt.matshow(df.corr(), fignum=f.number)
plt.xticks(range(df.select_dtypes(['number']).shape[1]), df.select_dtypes(['number']).columns, fontsize=10, rotation=90)
plt.yticks(range(df.select_dtypes(['number']).shape[1]), df.select_dtypes(['number']).columns, fontsize=10)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
plt.title('Correlation Matrix', fontsize=16);
```



Problem Identification

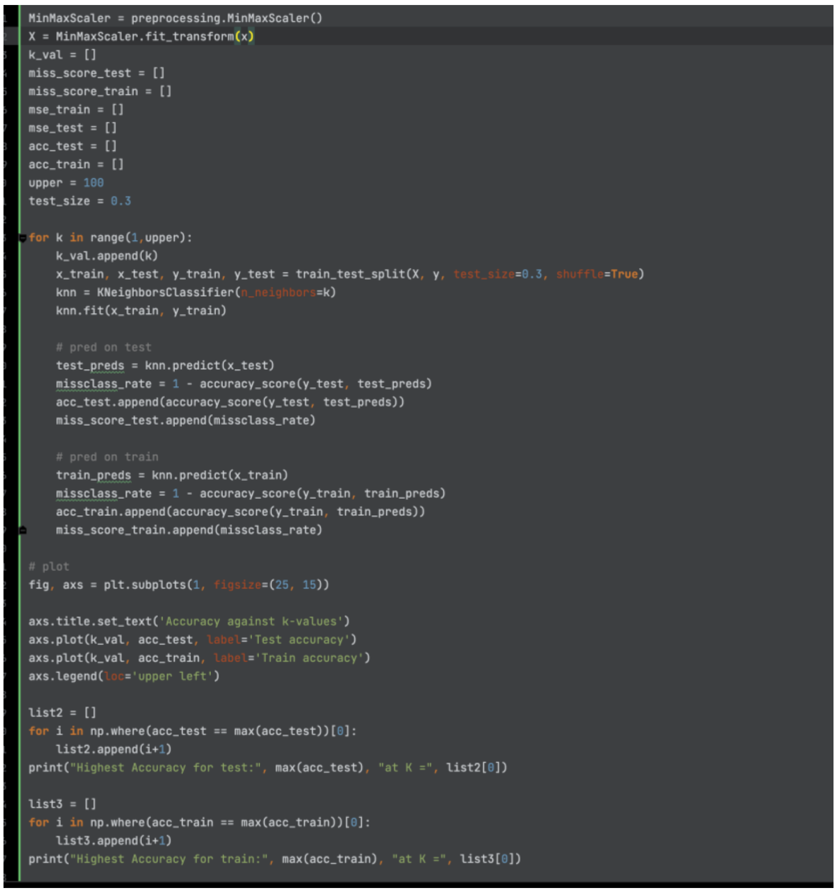
Plot number of starch present and starch absent in the dataset

```
import matplotlib.pyplot as plt
import seaborn as sns
df.loc[df['Starch \n(g)'] > 0, 'Classifier'] = 'Starch Present'
df.loc[df['Starch \n(g)'] == 0, 'Classifier'] = 'Starch Absent'

plt.figure(figsize=(7,5))
sns.countplot(x="Classifier", data=df)
plt.show()
```

Drop Unique Identifiers:

```
unique_identifiers = ['Public Food Key', 'Food Name', 'Classification'];
df = df.drop(columns=unique_identifiers);
print(df)
```

KNN average runs:

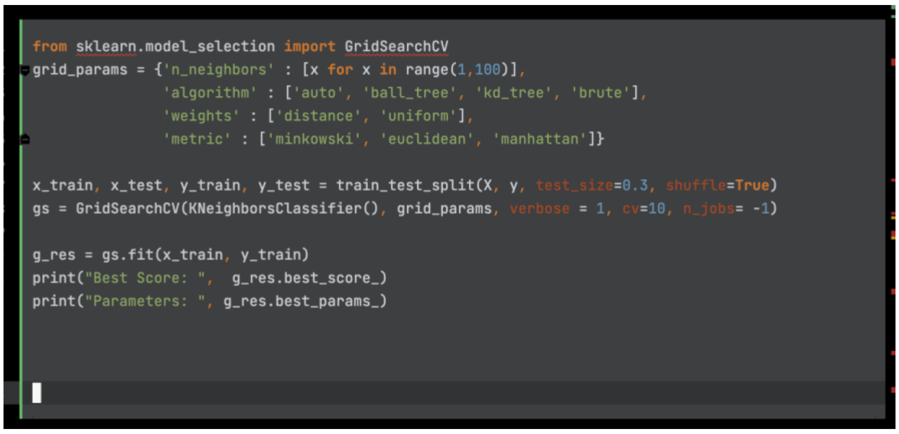


```python
MinMaxScaler = preprocessing.MinMaxScaler()
X = MinMaxScaler.fit_transform(x)
k_val = []
miss_score_test = []
miss_score_train = []
mse_train = []
mse_test = []
acc_test = []
acc_train = []
upper = 100
test_size = 0.3

for k in range(1,upper):
    k_val.append(k)
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=True)
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)

    # pred on test
    test_preds = knn.predict(x_test)
    missclass_rate = 1 - accuracy_score(y_test, test_preds)
    acc_test.append(accuracy_score(y_test, test_preds))
    miss_score_test.append(missclass_rate)

    # pred on train
    train_preds = knn.predict(x_train)
    missclass_rate = 1 - accuracy_score(y_train, train_preds)
    acc_train.append(accuracy_score(y_train, train_preds))
    miss_score_train.append(missclass_rate)

# plot
fig, axs = plt.subplots(1, figsize=(25, 15))

axs.title.set_text('Accuracy against k-values')
axs.plot(k_val, acc_test, label='Test accuracy')
axs.plot(k_val, acc_train, label='Train accuracy')
axs.legend(loc='upper left')

list2 = []
for i in np.where(acc_test == max(acc_test))[0]:
    list2.append(i+1)
print("Highest Accuracy for test:", max(acc_test), "at K =", list2[0])

list3 = []
for i in np.where(acc_train == max(acc_train))[0]:
    list3.append(i+1)
print("Highest Accuracy for train:", max(acc_train), "at K =", list3[0])
```

Cross Validation for KNN

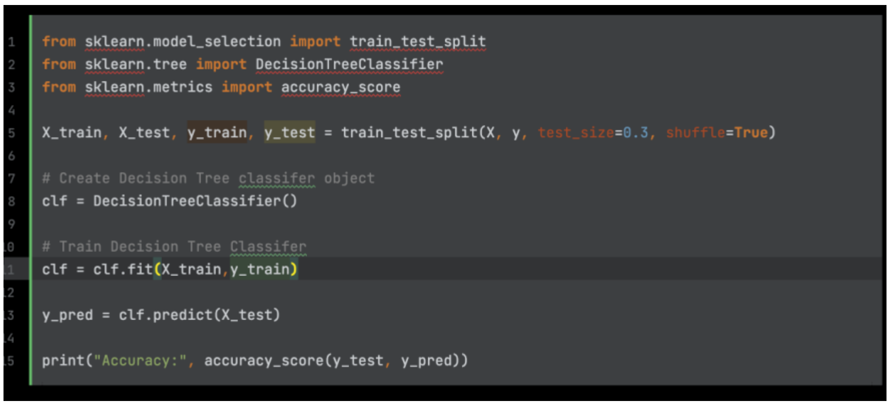
```python
from sklearn.model_selection import cross_val_score


# Perform 10-fold cross-validation
scores = cross_val_score(knn, X, y, cv=10)

print('Cross-validation scores:', scores)
print('Average cross-validation score:', scores.mean())
```

Grid Search for KNN



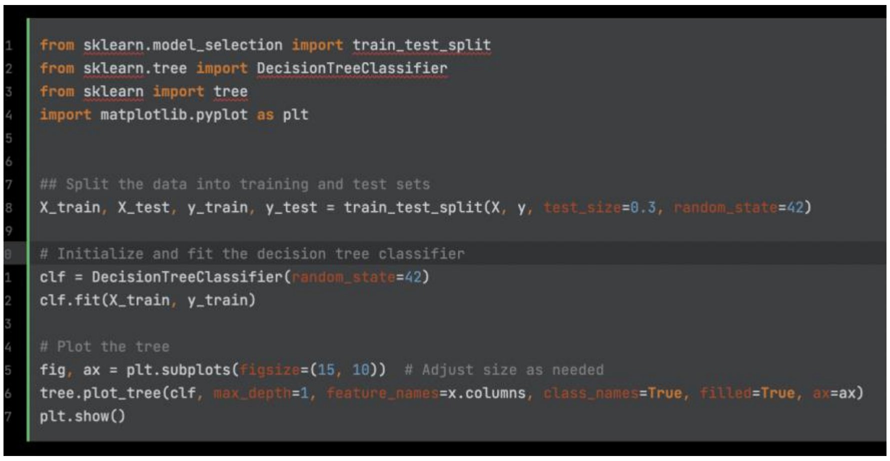
```python
from sklearn.model_selection import GridSearchCV
grid_params = {'n_neighbors' : [x for x in range(1,100)],
               'algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute'],
               'weights' : ['distance', 'uniform'],
               'metric' : ['minkowski', 'euclidean', 'manhattan']}

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=True)
gs = GridSearchCV(KNeighborsClassifier(), grid_params, verbose = 1, cv=10, n_jobs= -1)

g_res = gs.fit(x_train, y_train)
print("Best Score: ",  g_res.best_score_)
print("Parameters: ", g_res.best_params_)
```

Decision Tree Classifier:



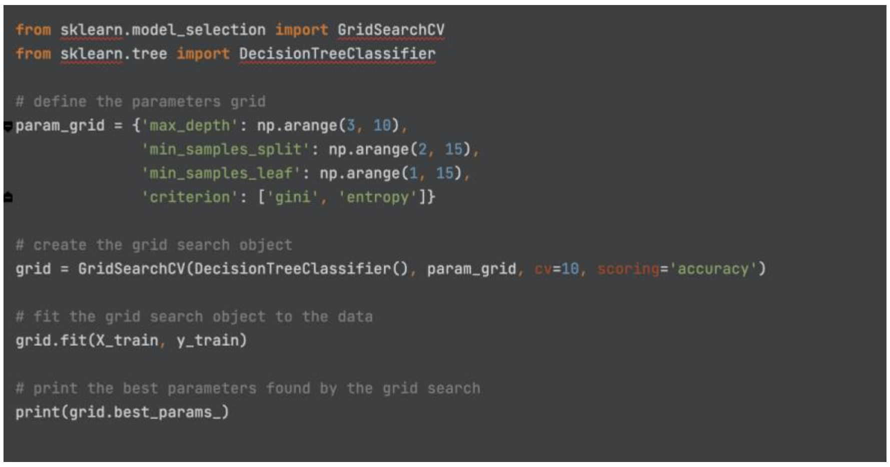
```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=True)

# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
```

Plot Decision Tree:



```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt


## Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and fit the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Plot the tree
fig, ax = plt.subplots(figsize=(15, 10))  # Adjust size as needed
tree.plot_tree(clf, max_depth=1, feature_names=X.columns, class_names=True, filled=True, ax=ax)
plt.show()
```

GridSearch for Decision Tree:

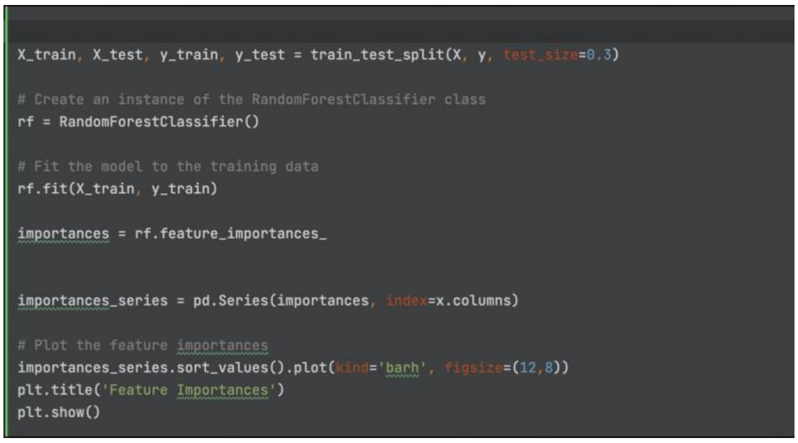


```python
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

# define the parameters grid
param_grid = {'max_depth': np.arange(3, 10),
              'min_samples_split': np.arange(2, 15),
              'min_samples_leaf': np.arange(1, 15),
              'criterion': ['gini', 'entropy']}

# create the grid search object
grid = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=10, scoring='accuracy')

# fit the grid search object to the data
grid.fit(X_train, y_train)

# print the best parameters found by the grid search
print(grid.best_params_)
```

Random Forest Classifier:

```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

accuracy_scores = []

# Run the random forest classifier 10 times
for i in range(10):
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

    # Create an instance of the RandomForestClassifier class
    rf = RandomForestClassifier(random_state=42)

    # Fit the model to the training data
    rf.fit(X_train, y_train)
    # Use the model to make predictions on the test data
    y_pred = rf.predict(X_test)

    # Calculate the accuracy of the model and add it to the list
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)

# Plot the accuracy scores

print(sum(accuracy_scores)/len(accuracy_scores))

plt.plot(range(1, 11), accuracy_scores)
plt.title('Accuracy scores of the random forest model over 10 runs')
plt.xlabel('Run number')
plt.ylabel('Accuracy score')
plt.show()
```

Least Important Features:



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Create an instance of the RandomForestClassifier class
rf = RandomForestClassifier()

# Fit the model to the training data
rf.fit(X_train, y_train)

importances = rf.feature_importances_


importances_series = pd.Series(importances, index=x.columns)

# Plot the feature importances
importances_series.sort_values().plot(kind='barh', figsize=(12,8))
plt.title('Feature Importances')
plt.show()
```

Grid Search on Random Forests Classifier:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Define the parameter grid for grid search
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of trees in the forest
    'max_depth': [None, 5, 10],  # Maximum depth of each tree
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],  # Minimum number of samples required to be at a leaf node
}

# Create an instance of the Random Forest classifier
rf = RandomForestClassifier()

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best parameters found by grid search
best_params = grid_search.best_params_

# Use the best model from grid search to make predictions on the test set
y_pred = grid_search.predict(X_test)

# Calculate the accuracy of the best model
accuracy = accuracy_score(y_test, y_pred)

print("Best parameters found by grid search:", best_params)
print("Accuracy of the best model:", accuracy)
```