Final Exam

by Jianwei Li

FILE	FINAL_EXAM.PDF (392.11K)		
TIME SUBMITTED	17-JUN-2021 11:07AM (UTC+1000)	WORD COUNT	936
SUBMISSION ID	1607743773	CHARACTER COUNT	4769

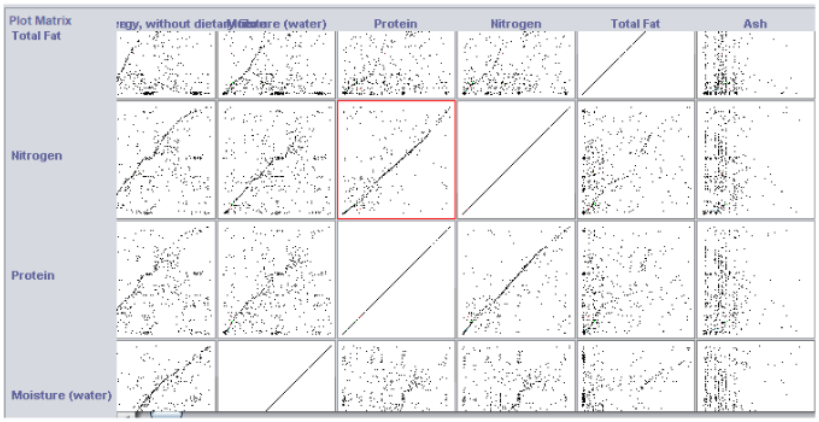
Data Preprocessing

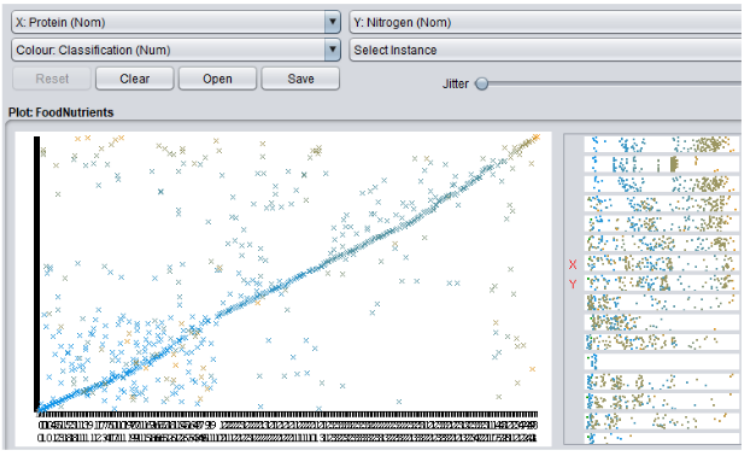
Since some of the data in 'FoodNutrients.csv' have errors, so we need to use Python to convert these data into correct format.

Code show as below:

```
import pandas as pd

# dataframe[column (series)] [row (Series index)]
data = pd.read_csv('FoodNutrients.csv')

data.to_csv('FoodNutrients.csv', index=False)
print ('done')
```

Bring into WEKA the data file 'FoodNutrients.csv', use 'Visualize' to generate a Plot Matrix.





It can be seen from the figure that 'Protein' and 'Nitrogen' may have a linear relationship.

Next we are going to conduct a polynomial regression and try to find a model that fits the two-dimensional data better.

Polynomial Regression

Data Preprocessing:

Open Matlab and import data from 'FoodNutrients.csv'.

Only need to import the two columns of data of attributes 'Protein' and 'Nitrogen', among which the output type is numeric matrix, exclude header, and rename the imported data to 'data_pr'.

Next delete rows in the 'data_pr' that contain missing data or NaN value.

Randomly divide the 'data_pr' into two parts: training set and test set. The training set accounts for 70% of the 'data_pr' and the test set accounts for 30%.

Code show as below:

```
% Delete rows in the data_set that contain missing data
data_pr = rmmissing(data_pr);
% Randomly divide the data set into 70% training set
% and 30% validation set
sz = size(data_pr);
col_sum = sz(1, 1);
col_training = round(col_sum*0.7);
row_idx_T = randperm(col_sum, col_training); % 70%
row_idx_V = [];

for i = 1:col_sum
    if ~ismember(i, row_idx_T)
        row_idx_V = [row_idx_V, i];
    end
end

training = data_pr(row_idx_T, :);
validation = data_pr(row_idx_V, :);

% Assign independent variables and response variables
x_T = training(:, 1); % Protein
y_T = training(:, 2); % Nitrogen
x_V = validation(:, 1);
y_V = validation(:, 2);

x_max = max(x_T);
x_min = min(x_T);
y_max = max(y_T);
y_min = min(y_T);

x = linspace(x_min, x_max);
```

Model Selection Procedure and Visualization:

Use cross-validation to find the optimal model complexity, train candidate models of different complexities on the training set and test their error on the validation set left out during training. Use the *polyfit*() and *polyval*() functions to produce a plot of training data set and fitted polynomials

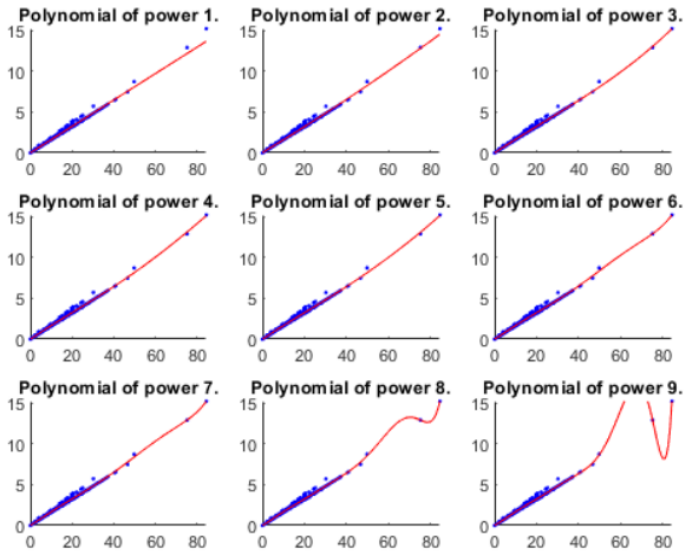
and a plot of error as a function of model order up to 9. For each order of polynomial model, use *polyfit*() and the training set to train the polynomial function so that it fits the training data, use *polyval*() to calculate the training and validation errors as a function of the polynomial order.

Code show as below:

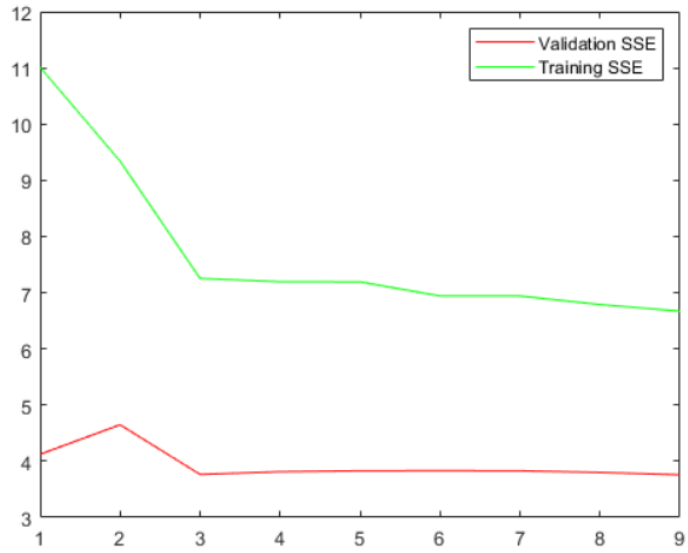
```
%Create figure of Data and fitted polynomials. To see clearly,
% break it into 9 subplots.
%For each subplot, plot training data set and fitted polynomial.
figure
trainError = zeros(9,col_training);
valError = zeros(9,col_sum-col_training);
trainSSE = zeros(9,1);
valSSE = zeros(9,1);

for R = 1:9
subplot(3,3,R);
hold on;

%Plot training data.
scatter(x_T, y_T, ".b");
% scatter(x_V, y_V, ".g");

%Create polynomial function that tries to fit training data.
poly = polyfit(x_T, y_T, R);

%Calculate y value at each x value, using polynomial function.
yfitted = polyval(poly, x);
plot(x, yfitted, 'r');

xlim([x_min, x_max]);
ylim([y_min, y_max]);
title(sprintf('Polynomial of power %d.', R));
hold off;

trainError(R,:) = y_T - polyval(poly, x_T);
valError(R,:) = y_V - polyval(poly,x_V);
trainSSE(R) = sum(trainError(R,:).^2);
valSSE(R) = sum(valError(R,:).^2);

end
```

```
%Plot figure of Error vs. polynomial order
figure
plot(1:9,valSSE,'r',1:9,trainSSE,'g');
legend('Validation SSE','Training SSE');
```

Data and fitted polynomials



Error vs. polynomial order



As can be seen from the figure above, the elbow is at order 3, as the model complexity increases, training error keeps decreasing before order 3, after that then stops decreasing or does not decrease further significantly, validation error firstly increases then decreases up to order 3 then stops decreasing. Therefore order 3 is the optimal model complexity, orders less than 3 are underfitting and orders greater than 3 are overfitting.

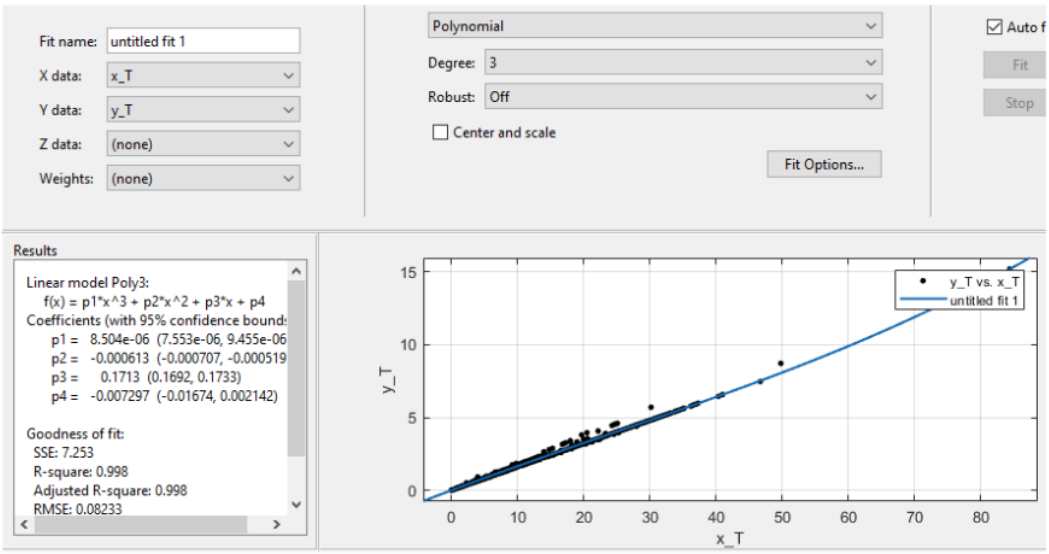
Of course, this phenomenon is also in line with the laws of nature. Protein contains nitrogen. With the increase of protein in food, nitrogen also increases, but the protein content in food will not increase indefinitely, as the content reaches a certain standard, increasing speed will gradually level off.

By opening the *cftool* and setting the degree to 3, we get the following polynomial equation:

$$f(x) = (8.504e^{-06})x^3 + (-0.000613)x^2 + 0.1713x - 0.007297$$

We can use this equation to fit or predict the data.

The output of cftool is as below:



Next, we will use Principal Component Analysis to explore the distribution of data.

Principal Component Analysis

Data Preprocessing:

Open Matlab and import data from 'FoodNutrients.csv'.

Only need to import the columns from 2 to 12, rows from 574 to 884, among which the output type is numeric matrix, exclude header, and rename the imported data to 'data_pca'.

Next delete rows in the 'data_pca' that contain missing data or NaN value.

Extract the first column('Classification') of data in 'data_pca' as a label, named 'label_pca', and the remaining columns of data as a training set, named 'training_pca'.

```
data_pca = data_pca(574:884, :);
data_pca = rmmissing(data_pca);
% Extract the label and training data set.
label_pca = data_pca(:, 1);
training_pca = data_pca(:, 2:end);
```

Training and Visualization:

Use matlab's built-in function *pca*() and training set 'training_pca' to train the model.

Calculate the percentage of the data variance accounted for by the first two principal components.

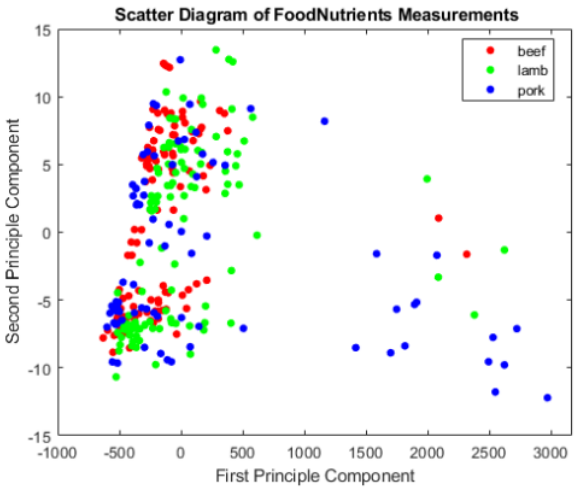
Produce a plot of the data in the space spanned by the first two principal components. Colour each point by its class.

Code show as below:

```
% Train PCA model
[coeff,score,latent] = pca(training_pca);
variance_proportion = cumsum(latent/sum(latent));
disp(variance_proportion);
```

```
% Plot the data in the space spanned by the first two principal
% components. Colour each point by its class.
score2 = score(:, 1:2);
figure
gscatter(score2(:, 1), score2(:,2), label_pca);
ax = gca; % current axes
lims = [ax.XLim ax.YLim]; % Extract the x and y axis limits
title('{\bf Scatter Diagram of FoodNutrients Measurements}');
xlabel('First Principle Component');
ylabel('Second Principle Component');
legend('beef', 'lamb','pork');
```

```
0.9999
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
```



Proportion of Variance

The percentage of the data variance accounted for by the first two principal components is 100%, this shows that the first two principal components summarizes all the attributes of the data.

In addition, it can be seen from the figure that the classification status of the data category after the data is mapped from the high-dimensional space to the two-dimensional space is non-linear separable. The distribution of the three types of meat is intertwined with each other, and there is no obvious classification boundary.

Next we will use the Support Vector Machine to train the data classification model.

Support Vector Machine

Data Preprocessing:

We will use the first two principal components ('score2') generated by PCA as the data set, and 'label_pca' as the identifier to label the classification of the data, rename it to 'label_svm'.

Trainning and Validation:

We will use MATLAB's built-in function *fitcecoc* () to train the SVM classification model for the data set 'score2'. Use the 10-fold cross-validation method to cross-validate the SVM classifier, and because the data presents a spherical distribution, the kernel function uses Radial Basis Function (**RBF**) kernel, optimization routine uses Sequence Minimum Optimization (**SMO**) method.

Then output the performance of the SVM classifier.

```
% convert numeric label to nominal
label_svm = nominal(label_pca);
```

```
% Control random number generation. For reproducibility
rng(1);
% Create an SVM template, and standardize the predictors
t = templateSVM('Standardize',true,"KernelFunction","rbf",'Solver','SMO');
% SVM classifier
SVMModel = fitcecoc(score2,label_svm,'Learners',t,...
    'ClassNames',{'18101','18102','18103'});
% to perform 10-fold cross-validation
CVSVMModel = crossval(SVMModel);
% Estimate the generalized classification error
classLoss = kfoldLoss(CVSVMModel);
disp('SVM Classifier Loss:');
disp(classLoss);
performance = 1-classLoss;
disp('SVM Classifier Performance:');
disp(performance);
```

```
SVM Classifier Loss:
    0.5016

SVM Classifier Performance:
    0.4984
```

The generalized classification error is 50%, which indicates that the SVM classifier generalizes fairly unwell. The reason for this phenomenon may be that it is difficult to infer which type of meat is beef, lamb or pork only by the content of nutrients.

This conclusion is also closer to the situation shown. Although the three types of meat come from different animals, these animals are all herbivorous mammals. Their meat quality and nutrients are very close, and there is no obvious distinction between them. This leads to the raw data of these

three types of meat are very close. Our SVM classifier was not well trained, so the classification performance is relatively low.