MECH 305

Report

Using a Binary Classifier to Predict if it will Rain in Australia

Osman Baalbaki

61121919

March 27th, 2020

1.0 Framing the Problem

Meteorology is one of the areas of science which most heavily depend on predictions and forecasts. In order to make accurate sophisticated predictions, meteorologists often rely on historical data and utilize machine learning algorithms to assess this data. For this assignment, I will be using historical weather data from Australia to predict rainfall. (i.e I will be using weather data from one day to predict if it will rain the following day). To focus the scope of the model, I will focus on one Australian City, Albury. The first 3012 rows in the dataset belong to this city.

2.0 Importing and Preprocessing

First, we start by preprocessing our data in Excel. Looking at the Cloud9AM and Cloud3PM, we notice that a significant portion of those columns is missing (NA). Therefore, these variables will likely not be very useful predictions, and to cleanup the data, I decide to remove them. Next, I decide to remove all rows where any of the continuous variables are "NA". Ideally this should be done only for our two chosen predictors as to not lose valuable data. But at this point, I still have not decided the two predictors and I will do it for all variables just for simplicity sake. (Note that this step does not remove many data points and we already have a big dataset, so it isn't that bad of an idea here). After Excel preprocessing is done, we are left with 2975 entries.

Next, the data for Albury is imported from the excel file into MATLAB, the data for other Australian cities in the dataset won't be relevant here as we are doing the analysis for Albury. We use a readtable function to see all the variables/column titles in the dataset. Below is the list of all variables and their type (continuous or categorical):

- Categorical
    - WindGustDir
    - WindDir9am
    - WindDir3pm
    - RainToday
    - RainTomrrow (Variable I am trying to predict)
- Continuous
    - MinTemp
    - MaxTemp
    - WindGustSpeed
    - WindSpeed9am
    - WindSpeed3pm
    - Humidity9am
    - Humidity3pm
    - Pressure9am
    - Pressure3pm
    - Temp9am
    - Temp3pm
    - RISK_MM

The next step is to determine which two features we will use in our classifier, since we are restricted to two by the tutorial instructions. For simplicity and to avoid encoding, I chose to eliminate all the categorical variables. Moreover, it can be argued that categorical variables provide less information to our model as they can only be one of few discrete values. In addition, since the dataset has four pairs of data of the same measurement taken at different times (Windspeed, Humidity, Temp, and Pressure at 9AM and 3PM), I decide to introduce four more variables that represent the difference in each pair. (i.e: DeltaHumidity, DeltaWindspeed, DeltaTemp, and DeltaPressure).

Following that, I used a Logistic Regression to determine which one of the available variables is the best predictor. After using a logistic fit, I will display the coefficient vector and search for the variables which have the biggest coefficients. However, when I attempted to run logistic regression with all our candidates MATLAB throws errors. Reducing the number of variables to 6 at a time allows the fit to work and I get a coefficient Matrix. Therefore, I split my variables into two groups, perform logistic regression on both groups, and take the variable beside the biggest coefficient from each group. Table 1 below shows the coefficients. (Note: all those variables are for the day prior to the day we are predicting if it is going to rain).

Group 1 Variables	Coefficient	Group 2 Variables	Coefficient
Minimum Temp	-0.0603	Delta Wind Speed	-0.012
Maximum Temp	-0.0296	Humidity 3 pm	-0.0783
Rainfall Amount	0.0099	Pressure 9 am	0.1098
Wind Gust Speed	-0.0715	Pressure 3 pm	0.0361
Wind Speed 9 am	0.0266	Delta Pressure	0.481
Wind Speed 3 pm	-0.0017	Temp 9 am	8.832
Delta Humidity	-0.0998	Temp 3 pm	-8.8658
Humidity 9 am	-0.0941	Delta Temp	9.0133

Therefore, for my classifiers I will be using the Delta Humidity and Delta Temperature variables.

Note: the issue of 'too many variables to analyze' reveals the importance of prior knowledge when preprocessing data. If I had sufficient knowledge about meteorology, I would've been able to eliminate variables that are irrelevant to predicting rainfall at the beginning of the preprocessing stage.

3.0 Running the Classifiers

For this assignment, I decided to try 7 different variations of classifiers. Those are:

- LDA
- Naïve Bayes
- KNN, with K =5
- KNN, with K = 50
- KNN, with K = 100
- SMV Linear
- SMV Polynomial

Those classifiers were all implemented in MATLAB, and the code is displayed in Appendix B. Next, the cross validation error was found using the kfoldLoss() function, and the CV error for all the classifiers is shown in the table below

ModelName	Error
"LDA"	0.14706
"Naive Bayes"	0.14916
"KNN, K = 5"	0.16345
"KNN, K = 50"	0.14664
"KNN, K = 100"	0.14748
"SVM-Linear"	0.15462
"SVM-Polynomial"	0.15504

As seen above, the CV error for all the classifiers used were close together and hovered around 0.15. This is a fairly low error and rectifies our choice of predictors. The model which has the lowest error is KNN with a K = 50, while LDA comes in a close second. It is not very conclusive if one of the methods is absolutely better than the others in this case, and further exploration could be needed. One way to further explore this, we can run a for-loop that tries K values from 1-100 in increments of one. I will not be doing this for this tutorial, as it is very computationally expensive. Overall, a visualization of the data may be necessary before we make a final decision for our model.

4.0 Visualizing the Boundaries

Having a visual picture of our classifiers' boundaries can give us a more intuitive idea on how well each classifier is performing and if there is any overfitting happening.

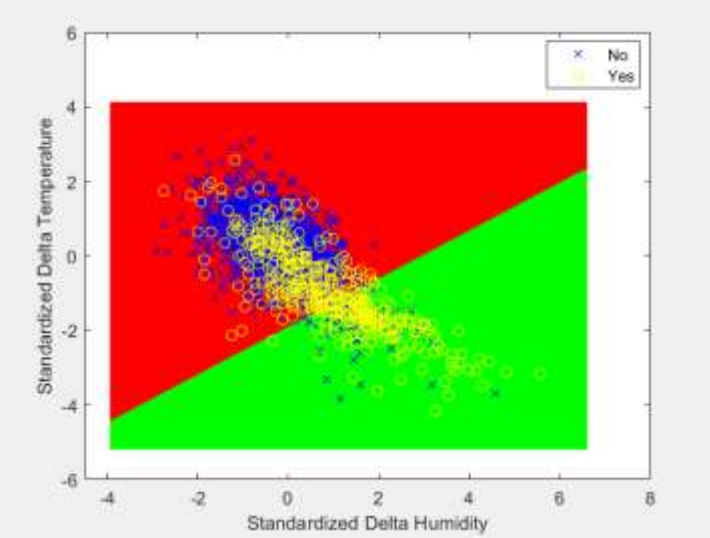


Figure 1: Boundary Layer Visualization for LDA

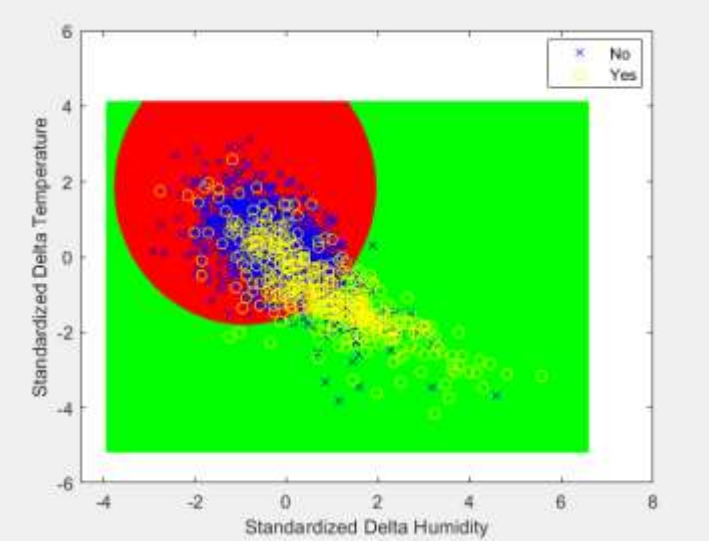


Figure 2: Boundary Layer Visualization for Naïve Bayes

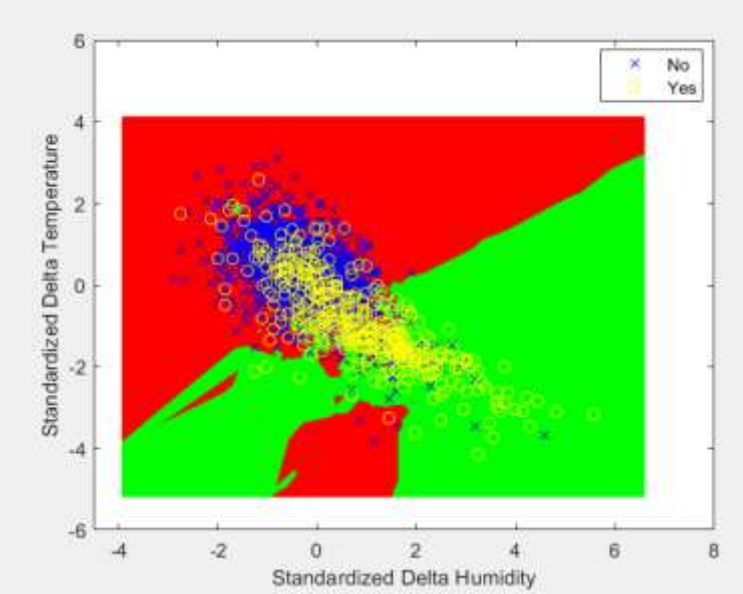


Figure 3: Boundary Layer Visualization for KNN with k =5

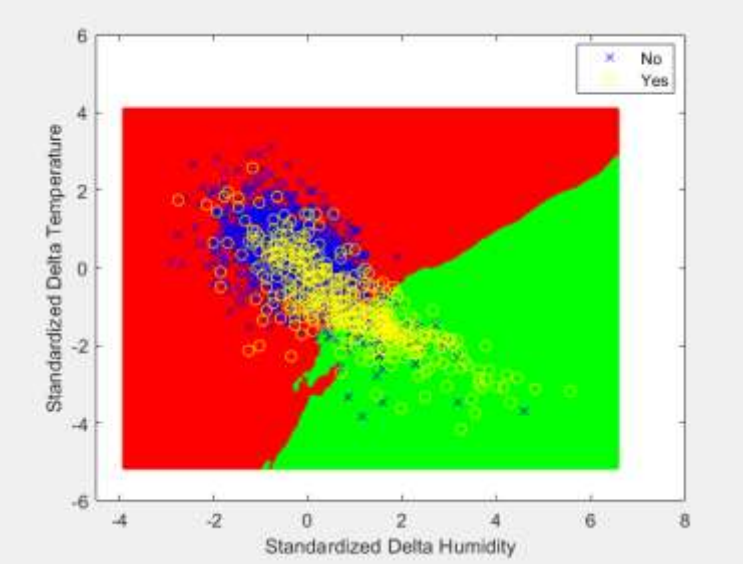


Figure 4: Boundary Layer Visualization for KNN with k = 50

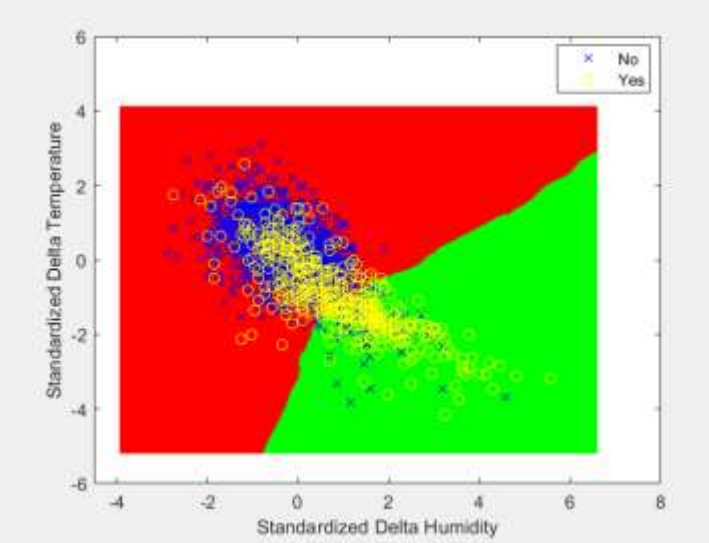


Figure 5: Boundary Layer Visualization for KNN with k 100

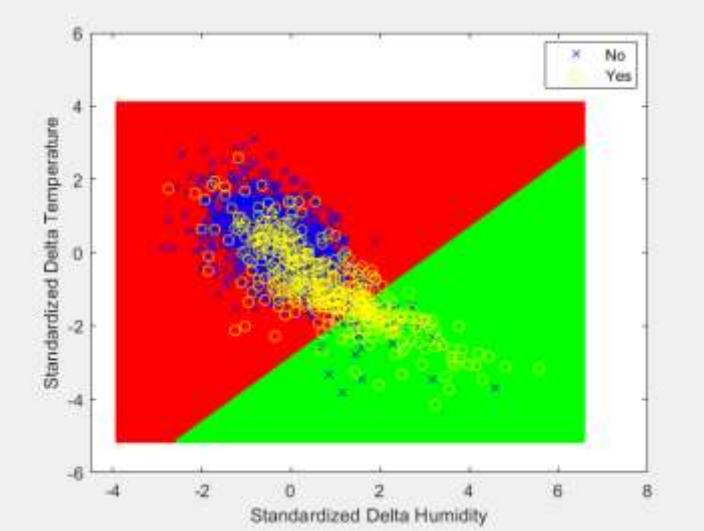


Figure 6: Boundary Layer Visualization for SVM- Linear

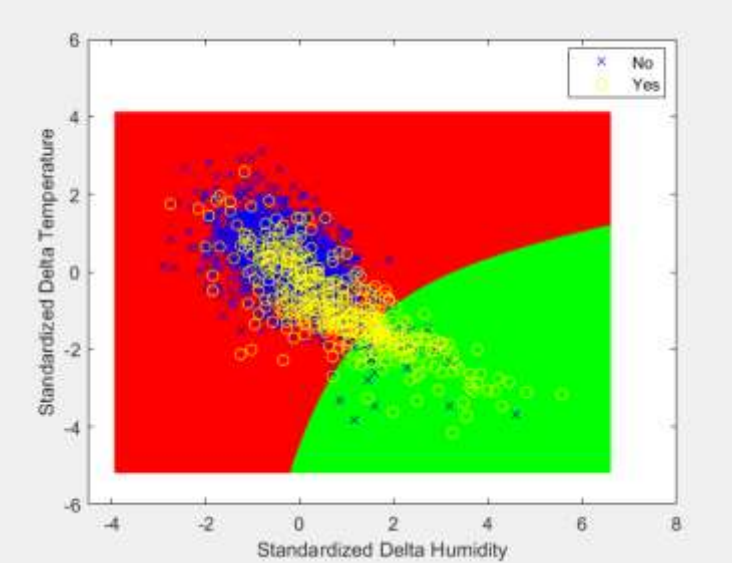


Figure 7: Boundary Layer Visualization for SVM-Polynomial

Looking at these visualizations, one observation is that the KNN classifier with k=5 overfits the data. This is represented by the red cluster that is separated from the rest of the red area. This is expected as KNNs with small k value tend to overfit data. The KNN with k = 50 also has a minor tendency to overfit. Overall, most of the classifiers have similar performance, and therefore I choose to test the generalization error for LDA and KNN with k=50 and k =100 (classifiers with three lowest CV error).

5.0 Assessing the chosen classifier

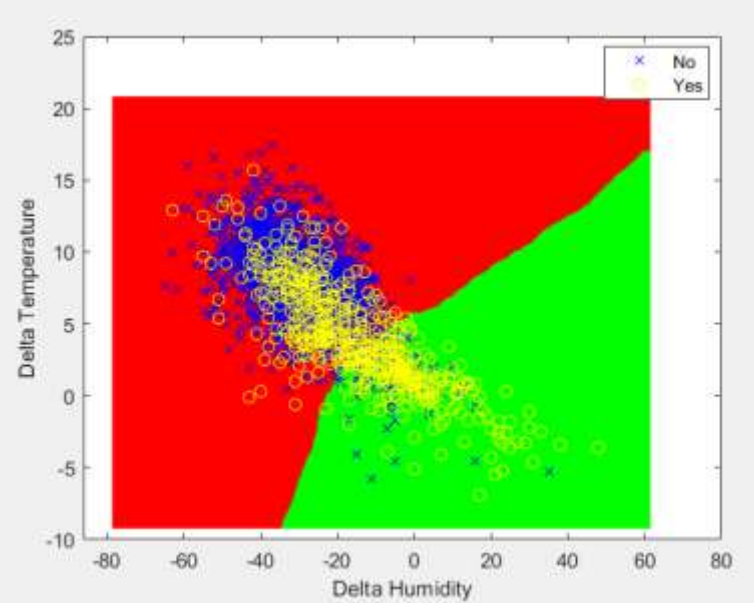
To assess the performance of the classifier, we find the generalization error. This error evaluates how the data performs on the test data, or data it hasn't seen before. This means that overfitting will not result in a higher accuracy here and this kind of error is a very good indicator on how our model will perform in a real-life application. Running the last part of the code shown in Appendix B, we find that:

Classifier	**Generalization Error**
KNN, with k = 50	0.1431
KNN, with k = 100	0.138
LDA	0.1448

We find that a KNN with k = 100 has the lowest generalization error.

6.0 Presenting the Solution

To tackle the problem of predicting whether it will rain in Albury, I propose using a KNN classifier with k = 100. Using this classifier, I was able to predict if it will rain tomorrow with an 86.2% accuracy. The only data I need for this prediction is Delta Humidity (Humidity at 3 pm – Humidity at 9 am) and Delta Temperature (Temperature at 3 pm – Temperature at 9 am) for the previous day. Note that many of the other factors we have available for the previous day, such as Wind Speed at 3 pm and Delta Pressure are not critical factors, and it is sufficient for us to just collect data about Delta Humidity and Delta Temp in the future.



7.0 Appendix A: Using Logistic Regression to select predictors

```
%Getting Data
data = readtable('Tut8.csv');
data.RainTomorrow = categorical(data.RainTomorrow);
Y = data.RainTomorrow;
X1 = data(:,{'MinTemp', 'MaxTemp', 'Rainfall',
'WindGustSpeed','WindSpeed9am', 'WindSpeed3pm',
'DeltaHumidity', 'Humidity9am'});
X2 =
data(:,{'DeltaWindSpeed','Humidity3pm','Pressure9am','P
ressure3pm','DeltaPressure','Temp9am','Temp3pm','DeltaT
emp'});
normalize(X1);
[B1,dev1,stats1] = mnrfit(X1{:,:},Y);
B1
normalize(X2);
[B2,dev2,stats2] = mnrfit(X2{:,:},Y);
B2
```

8.0 Appendix B: Classifier code

```
%Mohamad Osman Baalbaki
%Tutorial 8

%Getting Data
data = readtable('Tut8.csv');
data.RainTomorrow = categorical(data.RainTomorrow);


%Train-Test Split
cv = cvpartition(height(data), 'HoldOut', 0.2);
train_data = data(cv.training,:);
test_data = data(cv.test,:);



%Setting up the features and target. Note we already determined
%best features in the other script using logistic regression
X = train_data(:, {'DeltaHumidity', 'DeltaTemp'});
m_train = height(X);
X = normalize(X);
Y = train_data.RainTomorrow;

%Visualizing
%figure(10)
%gscatter(X.var1, X.var2, Y)
%xlabel('var1')
%ylabel('var2')

%LDA
da_clf = fitcdiscr(X, Y);
clf_cv = crossval(da_clf);
LDA_cv_error = kfoldLoss(clf_cv);


%Naive Bayes
nb_clf = fitcnb(X, Y);
nb_clf_cv = crossval(nb_clf);
nb_cv_error = kfoldLoss(nb_clf_cv);


%K-NN  with K = 5
knn5_clf = fitcknn(X, Y, 'NumNeighbors',5);
knn5_clf_cv = crossval(knn5_clf);
knn5_cv_error = kfoldLoss(knn5_clf_cv);


%K-NN  with K = 50
knn50_clf = fitcknn(X, Y, 'NumNeighbors',50);
knn50_clf_cv = crossval(knn50_clf);
knn50_cv_error = kfoldLoss(knn50_clf_cv);



%K-NN  with K = 100
knn100_clf = fitcknn(X, Y, 'NumNeighbors',100);
knn100_clf_cv = crossval(knn100_clf);
knn100_cv_error = kfoldLoss(knn100_clf_cv);


%Linear SVM
svmln_clf = fitcsvm(X, Y);
```

```
svmln_clf_cv = crossval(svmln_clf);
SVML_cv_error = kfoldLoss(svmln_clf_cv);



%Poly SVM
svmPoly_clf = fitcsvm(X, Y, 'KernelFunction','polynomial');
svmPoly_clf_cv = crossval(svmPoly_clf);
SVMP_cv_error = kfoldLoss(svmPoly_clf_cv);



%Classifier table
cv_errors = table("LDA", [LDA_cv_error],'VariableNames',
{'ModelName','Error'});
cv_errors = [cv_errors; {"Naive Bayes", nb_cv_error}];
cv_errors = [cv_errors; {"KNN, K = 5", knn5_cv_error}];
cv_errors = [cv_errors; {"KNN, K = 50", knn50_cv_error}];
cv_errors = [cv_errors; {"KNN, K = 100", knn100_cv_error}];
cv_errors = [cv_errors; {"SVM-Linear", SVML_cv_error}];
cv_errors = [cv_errors; {"SVM-Polynomial", SVMP_cv_error}];



%Visualize Boundaries
figure(1)
Hum_range = min(X.DeltaHumidity)-1:0.01:max(X.DeltaHumidity)+1;
Temp_range = min(X.DeltaTemp)-1:0.01:max(X.DeltaTemp)+1;
[xx1, xx2] = meshgrid(Hum_range,Temp_range);
XGrid = [xx1(:) xx2(:) ];
predictions_meshgrid = predict(da_clf,XGrid);
gscatter(xx1(:), xx2(:), predictions_meshgrid,'rgb');
hold on
gscatter(X.DeltaHumidity, X.DeltaTemp, Y, 'by','xo')
xlabel('Standardized Delta Humidity')
ylabel('Standardized Delta Temperature')

figure(2)
predictions_meshgrid = predict(nb_clf,XGrid);
gscatter(xx1(:), xx2(:), predictions_meshgrid,'rgb');
hold on
gscatter(X.DeltaHumidity, X.DeltaTemp, Y, 'by','xo')
xlabel('Standardized Delta Humidity')
ylabel('Standardized Delta Temperature')



figure(3)
predictions_meshgrid = predict(knn5_clf,XGrid);
gscatter(xx1(:), xx2(:), predictions_meshgrid,'rgb');
hold on
gscatter(X.DeltaHumidity, X.DeltaTemp, Y, 'by','xo')
xlabel('Standardized Delta Humidity')
ylabel('Standardized Delta Temperature')

figure(4)
predictions_meshgrid = predict(knn50_clf,XGrid);
gscatter(xx1(:), xx2(:), predictions_meshgrid,'rgb');
hold on
gscatter(X.DeltaHumidity, X.DeltaTemp, Y, 'by','xo')
xlabel('Standardized Delta Humidity')
ylabel('Standardized Delta Temperature')
```

```
132.0
133.0
134.0  figure(5)
135.0  predictions_meshgrid = predict(knn100_clf,XGrid);
136.0  gscatter(xx1(:), xx2(:), predictions_meshgrid,'rgb');
137.0  hold on
138.0  gscatter(X.DeltaHumidity, X.DeltaTemp, Y, 'by','xo')
139.0  xlabel('Standardized Delta Humidity')
140.0  ylabel('Standardized Delta Temperature')
141.0
142.0
143.0  figure(6)
144.0  predictions_meshgrid = predict(svmln_clf,XGrid);
145.0  gscatter(xx1(:), xx2(:), predictions_meshgrid,'rgb');
146.0  hold on
147.0  gscatter(X.DeltaHumidity, X.DeltaTemp, Y, 'by','xo')
148.0  xlabel('Standardized Delta Humidity')
149.0  ylabel('Standardized Delta Temperature')
150.0
151.0
152.0  figure(7)
153.0  predictions_meshgrid = predict(svmPoly_clf,XGrid);
154.0  gscatter(xx1(:), xx2(:), predictions_meshgrid,'rgb');
155.0  hold on
156.0  gscatter(X.DeltaHumidity, X.DeltaTemp, Y, 'by','xo')
157.0  xlabel('Standardized Delta Humidity')
158.0  ylabel('Standardized Delta Temperature')
159.0
160.0
161.0  %Generelization Error
162.0
163.0  X_test = test_data(:, {'DeltaHumidity', 'DeltaTemp'});
164.0  X_test.DeltaHumidity = (X_test.DeltaHumidity -
       mean(train_data.DeltaHumidity))/std(train_data.DeltaHumidity);
165.0  X_test.DeltaTemp = (X_test.DeltaTemp -
       mean(train_data.DeltaTemp))/std(train_data.DeltaTemp);
166.0  Y_test_pred = predict(da_clf, X_test); % Select your clf here
167.0  accuracy = sum((Y_test_pred == test_data.RainTomorrow))/ height(test_data);
168.0  generalization_errorknn = 1 - accuracy
169.0
170.0
171.0
172.0  %Presenting Solution
173.0
174.0
175.0  figure(8)
176.0  predictions_meshgrid = predict(knn100_clf,XGrid);
177.0  % De-normalize grid data:
178.0  DeltaHumidity_grid = xx1(:)*std(train_data.DeltaHumidity) +
       mean(train_data.DeltaHumidity);
179.0  DeltaTemp_grid = xx2(:)*std(train_data.DeltaTemp) + mean(train_data.DeltaTemp);
180.0  % Draw decision boundary on the physical space:
181.0  gscatter(DeltaHumidity_grid,DeltaTemp_grid, predictions_meshgrid,'rgb');
182.0  hold on
183.0  % Draw the actual data points
184.0  gscatter(data.DeltaHumidity, data.DeltaTemp, data.RainTomorrow, 'by','xo')
185.0  xlabel('Delta Humidity')
186.0  ylabel('Delta Temperature')
187.0  %axis([min(data.DeltaHumidity)-5, max(data.DeltaHumidity)+5, ...
188.0  % min(data.DeltaTemp)-10000, max(data.DeltaTemp)+10000])
```