# Udacity Machine Learning Nanodegree 2020

## Capstone Report

## Childhood Autistic Spectrum Disorder Screening using Machine Learning

**Mukkul Kurwa**

**March 2020**

**Project Overview:** Autistic Spectrum Disorder (ASD) is the name for a group of developmental disorders impacting the nervous system. ASD symptoms range from mild to severe: mainly language impairment, challenges in social interaction, and repetitive behaviors. Many other possible symptoms include anxiety, mood disorders and Attention-Deceit/Hyperactivity Disorder (ADHD).

ASD has a significant economic impact in the healthcare domain, both due to the increase in the number of ASD cases, and because of the time and costs involved in diagnosing a patient. Early detection of ASD can help both patients and the healthcare sector by prescribing patients the therapy and/or medication they need and thereby reducing the long-term costs associated with delayed diagnosis. Thus, health care professionals across the globe have an urgent need for easy, time-scient, robust and accessible ASD screening methods that can accurately predict whether a patient with certain measured characteristics has ASD and inform individuals whether they should pursue formal clinical diagnosis.

However, challenges remain. Pursuing such research necessitates working with datasets that record information related to behavioral traits and other factors such as gender, age, ethnicity, etc. Such datasets are rare, making it di cult to perform thorough analyses to improve the efficiency, sensitivity, specie city and predictive accuracy of the ASD screening process. At present, very limited autism datasets associated with clinical or screening are available and most of them are genetic in nature. These data are extremely sensitive and hard to collect for social and personal reasons and the regulations around them.

**Origins of ASD Data Set:** I was able to use open source data available at UCI Machine Learning Repository. The data was made available to the public recently on December 24th, 2017. The data set, which I will be referring to as the ASD data set from here on out, came with a .csv le that contains 292 instances that are described by 21 attributes, a mix of numerical and categorical variables. A short description of ASD dataset can be found on this page. This data set was denoted by Prof.Fadi Fayez Thabtah, Department of Digital Technology, MIT, Auckland, New Zealand, fadi.fayez@manukau.ac.nz. Further details of the raw data follow in Section 2.

**Problem Statement:** The problem is to diagnose Autistic Spectrum Disorder (ASD) based on behavioral features and individual characteristics. This is a classification problem where the model takes behavioral features and individual characteristics and detects whether the patient has Autistic Spectrum Disorder (ASD) or not. I have used Keras to build and train our network. This model is relatively simple and will only use dense (also known as fully connected) layers. This is the most common neural network layer. The network has one hidden layer, uses an Adam optimizer, and a categorical cross entropy loss. Once the model is trained, I have to tested its performance on the testing dataset. The model has never seen this information before; as a result, the testing dataset allows me to determine whether or not the model is able to generalize

to information that wasn't used during its training phase. I have used some of the metrics provided by Scikit-learn for this purpose such as classification reports and accuracy score.

**Metrics:** Once the model is trained, I need to test its performance on the testing dataset. The model has never seen this information before; as a result, the testing dataset allows me to determine whether or not the model will be able to generalize to information that wasn't used during its training phase. I have used some of the metrics provided by Scikit-learn for this purpose such as classification reports and accuracy score.

In order to choose the appropriate model that avoids Underfitting or Overfitting the data we will analyze the Bias-Variance Trade-O, Model Complexity Graph, Learning Curves and Receiver Operator Characteristic Curves (ROC). To measure the effectiveness of each classification model we will study the accuracy score along with the precision, recall, F-Beta Score and confusion matrix.

**Definition: Model Complexity Graph**

Our goal is to always the model that will generalize well to unseen data. A model-complexity graph plots the training error and cross validation error as the model's complexity varies, i.e., the x-axis represents the complexity of the model (such as degree of the polynomial for regression or depth of a decision tree) and the y-axis measures the error in training and cross validation. Note that the size of the data set remains constant even while model complexity varies.

Figure (1) below shows a typical model complexity graph. On the left we have a model that under ts and we see high training and cross-validation errors, while on the right we have a model that over ts and gives us low training error but high cross validation error. The model in the middle is just right, with relatively low training and testing errors, and this is the model we should select. The best predictive and fitted model (neither underfitting nor overfitting) would be where the cross-validation error achieves its global minimum.

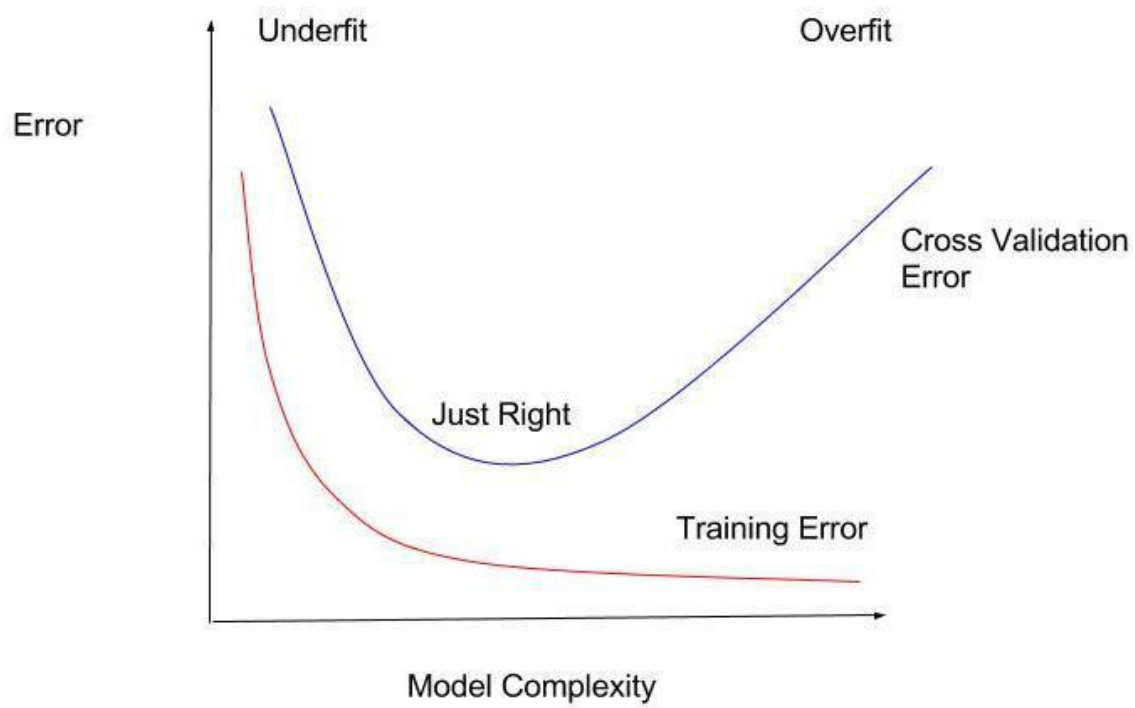


Figure 1: Model Complexity Graph

**Definition: Learning Curves**

Learning curves provide a way for us to distinguish between underfitting, overfitting, or the model that is just right. Learning curves are a plot of both the training error and cross validation error versus the size of the training set. In other words, a learning curve in machine learning is a graph that compares the errors of a model on training and cross validation data over a varying number of training instances. By separating training and cross validation sets and graphing errors of each, learning curves help us understand how well the model will potentially generalize to the unseen testing data.

Typically, we observe that the training error increases with the size of the training set, since we have more points to t the model to. A learning curve allows us to verify when a model has learned as much as it can about the data. When this occurs, both the training and cross validation errors reach a plateau and there is a consistent gap between the two error rates. In the case of underfitting or a high-bias model the two curves converge to a high point. In the perfect model the curves converge to a low point. In the case of overfitting or a high-variance model, the curves do not converge; the training curve stays low and the cross-validation error stays high, since models that overt tend to memorize the training data. (See Figure 2).

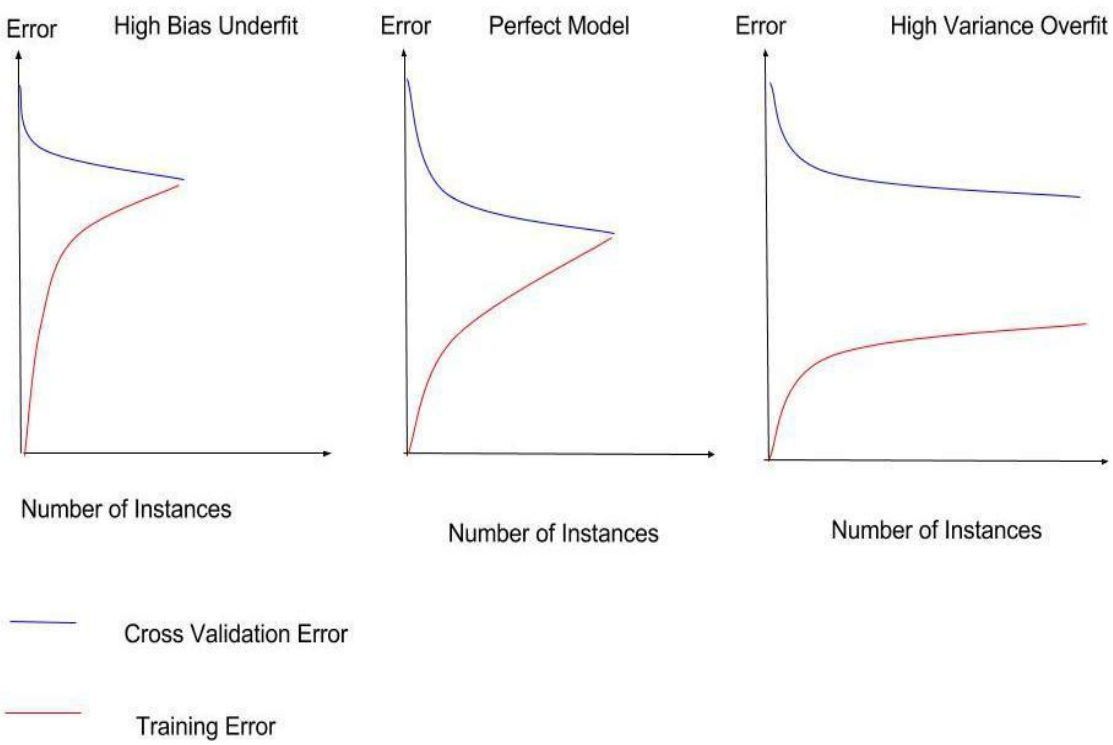



Figure 2: Learning Curves

**Definition: Receiver Operating Characteristic (ROC) Curves**

The receiver operating characteristic curve is a commonly used summary for assessing the diagnostic ability of a binary classifier over all possible thresholds. It is a plot of the true-positive rate (also known as sensitivity or recall) versus the false-positive rate (also known as specificity). The area under the ROC curve gives an idea about the overall performance of a classifier, summarized. The bigger the area under its curve, the better the overall performance of the binary classifier.

Given a set of labeled data and a predictive model, every data point will lie in one of the four categories:

True positive: The adult individual DID have ASD and we correctly predicted that the individual would have ASD.

True negative: The individual did NOT have ASD and we correctly predicted that the individual would NOT have ASD.

False positive (Type 1 Error): The individual did NOT have ASD, but we incorrectly predicted that the individual would have ASD.

False negative (Type 2 Error): The individual DID have ASD, but we incorrectly predicted that the individual would NOT have ASD.

These counts can also be represented in a confusion matrix (see Table 1) that allows us to visualize the performance of a supervised machine learning algorithm:

| | individual with ASD | individual with no ASD |
|---|---|---|
| predicted: ASD=‘YES’<br>predicted: ASD=‘NO’ | True Positive<br>False Negative | False Positive<br>True Negative |

Table 1: Confusion Matrix

**Definition: Accuracy**

Accuracy measures how often the classifier makes the correct prediction. In other words, it is the ratio of the number of correct predictions to the total number of predictions (the number of test data points). Accuracy is defined as the fraction of correct predictions.

$$\text{accuracy} = \frac{\text{true positive} + \text{true negative}}{\text{true positive} + \text{false positive} + \text{false negative} + \text{true negative}}$$

**Definition: Precision**

Precision measures how accurate our positive predictions were i.e., out of all the points predicted to be positive how many of them were actually positive.

$$\text{precision} = \frac{\text{true positive}}{\text{true positive+false positive}}$$

**Definition: Recall**

Recall measures what fraction of the positives our model identified, i.e., out of the points that are labelled positive, how many of them were correctly predicted as positive. Another way to think about this is what fraction of the positive points were my model able to catch?

$$\text{recall} = \frac{\text{true positive}}{\text{true positive+false negative}}$$

For classification problems that are skewed in their classi cation distributions like ours where we have a total of 609 records (after data preprocessing) with 180 individuals diagnosed with ASD and 429 individuals not diagnosed with ASD, accuracy by itself is not a very good metric. Thus, in this case precision and recall come in very handy. These two metrics can be combined to get the F1 score, which is weighted average (harmonic mean) of the precision and recall scores. This score can range from 0 to 1, with 1 being the best possible F1 score(we take the harmonic mean as we are dealing with ratios).

**Definition: F1 Score**

The F 1 Score is the harmonic mean of precision and recall and thus must lie between them. The F 1 Score is closer to the smaller of precision and recall than the higher number. As a result, if one of the precision or recall value is low the F 1 Score raises a ag.

$$F\ 1\ \ \text{Score} = 2\ \ \frac{\text{precision}\ \ \text{recall}}{\text{precision} + \text{recall}}$$

**Definition: F2 Score**

The F score is the weighted harmonic mean of precision and recall, reaching its optimal value at 1 and its worst value at 0. The parameter determines the weight of precision in the combined score, i.e., < 1 lends more weight to precision, and > 1 favors recall. Further, = 0 considers only precision and = 1 considers only recall.

**Data Exploration:**

Our data set involves ten behavioral features (\AQ-10-Child") (binary data) and ten individual characteristics such as Gender, Ethnicity, Age, etc. (categorical data) and one numerical data result below lists all variables involved in the ASD data set.

## Attribute Information:

Table 1: Features and their descriptions

| Attribute | Type | Description |
|---|---|---|
| Age | Number | years |
| Gender | String | Male or Female |
| Ethnicity | String | List of common ethnicities in text format |
| Born with jaundice | Boolean (yes or no) | Whether the case was born with jaundice |
| Family member with PDD | Boolean (yes or no) | Whether any immediate family member has a PDD |
| Who is completing the test | String | Parent, self, caregiver, medical staff, clinician ,etc. |
| Country of residence | String | List of countries in text format |
| Used the screening app before | Boolean (yes or no) | Whether the user has used a screening app |
| Screening Method Type | Integer (0,1,2,3) | The type of screening methods chosen based on age category (0=toddler, 1=child, 2= adolescent, 3= adult) |
| Question 1 Answer | Binary (0, 1) | The answer code of the question based on the screening method used |
| Question 2 Answer | Binary (0, 1) | The answer code of the question based on the screening method used |
| Question 3 Answer | Binary (0, 1) | The answer code of the question based on the screening method used |
| Question 4 Answer | Binary (0, 1) | The answer code of the question based on the screening method used |
| Question 5 Answer | Binary (0, 1) | The answer code of the question based on the screening method used |
| Question 6 Answer | Binary (0, 1) | The answer code of the question based on the screening method used |
| Question 7 Answer | Binary (0, 1) | The answer code of the question based on the screening method used |
| Question 8 Answer | Binary (0, 1) | The answer code of the question based on the screening method used |
| Question 9 Answer | Binary (0, 1) | The answer code of the question based on the screening method used |
| Question 10 Answer | Binary (0, 1) | The answer code of the question based on the screening method used |
| Screening Score | Integer | The final score obtained based on the scoring algorithm of the screening method used. This was computed in an automated manner |

Our raw data set contains 292 instances with 141 individuals diagnosed with ASD. Thus, the percentage of individuals diagnosed with ASD is 48.29%. Here is a sample of what the ASD data looks like:

```
[1] # Import libraries necessary for this project
    import numpy as np
    import pandas as pd
    from time import time
    from IPython.display import display # Allows the use of display() for DataFrames

    # Pretty display for notebooks
    %matplotlib inline

    data = pd.read_csv('Autism-Child-Data.csv')
    display(data.head(n=5))
```

| | id | A1_Score | A2_Score | A3_Score | A4_Score | A5_Score | A6_Score | A7_Score | A8_Score | A9_Score | A10_Score | age | gender | ethnicity | jundice | austim | contry_of_res | used_app_before | result | age_desc | relation | Class/ASD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 6 | m | Others | no | no | Jordan | no | 5 | 4-11 years | Parent | NO |
| 1 | 2 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 6 | m | Middle Eastern | no | no | Jordan | no | 5 | 4-11 years | Parent | NO |
| 2 | 3 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 6 | m | ? | no | no | Jordan | yes | 5 | 4-11 years | ? | NO |
| 3 | 4 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 5 | f | ? | yes | no | Jordan | no | 4 | 4-11 years | ? | NO |
| 4 | 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 5 | m | Others | yes | no | United States | no | 10 | 4-11 years | Parent | YES |

```
# Total number of records
n_records = len(data.index)

# TODO: Number of records where individual's with ASD
n_asd_yes = len(data[data['Class/ASD'] == 'YES'])

# TODO: Number of records where individual's with no ASD
n_asd_no = len(data[data['Class/ASD'] == 'NO'])

# TODO: Percentage of individuals whose are with ASD
yes_percent = float(n_asd_yes) / n_records *100

# Print the results
print("Total number of records: {}".format(n_records))
print("Individuals diagonised with ASD: {}".format(n_asd_yes))
print("Individuals not diagonised with ASD: {}".format(n_asd_no))
print("Percentage of individuals diagonised with ASD: {:.2f}%".format(yes_percent))
```

```
Total number of records: 292
Individuals diagonised with ASD: 141
Individuals not diagonised with ASD: 151
Percentage of individuals diagonised with ASD: 48.29%
```

Below will give the summary statistics for each variable:

```
[3] data.describe()
```

| | id | A1_Score | A2_Score | A3_Score | A4_Score | A5_Score | A6_Score | A7_Score | A8_Score | A9_Score | A10_Score | result |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 292.000000 | 292.000000 | 292.000000 | 292.000000 | 292.000000 | 292.000000 | 292.000000 | 292.000000 | 292.000000 | 292.000000 | 292.000000 | 292.000000 |
| mean | 146.500000 | 0.633562 | 0.534247 | 0.743151 | 0.551370 | 0.743151 | 0.712329 | 0.606164 | 0.496575 | 0.493151 | 0.726027 | 6.239726 |
| std | 84.437354 | 0.482658 | 0.499682 | 0.437646 | 0.498208 | 0.437646 | 0.453454 | 0.489438 | 0.500847 | 0.500811 | 0.446761 | 2.284882 |
| min | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 73.750000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 5.000000 |
| 50% | 146.500000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 6.000000 |
| 75% | 219.250000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 8.000000 |
| max | 292.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 10.000000 |

## Exploratory Visualization:

Before proceeding to apply any algorithm, we take a moment to visualize the ASD data set using the Seaborn module of Python.

We generate side-by-side box plots, which present various distribution of the feature ‘result’ with respect to ‘gender’ and ‘relation’. In Figure 5, the boxplots in red show the distributions of the data which belongs to the ‘ASD class’ whereas the blue one showing the distributions of the data which are non-autistic. This gives us our impression of the internal connections of some of the above-mentioned features that are present in our data set.

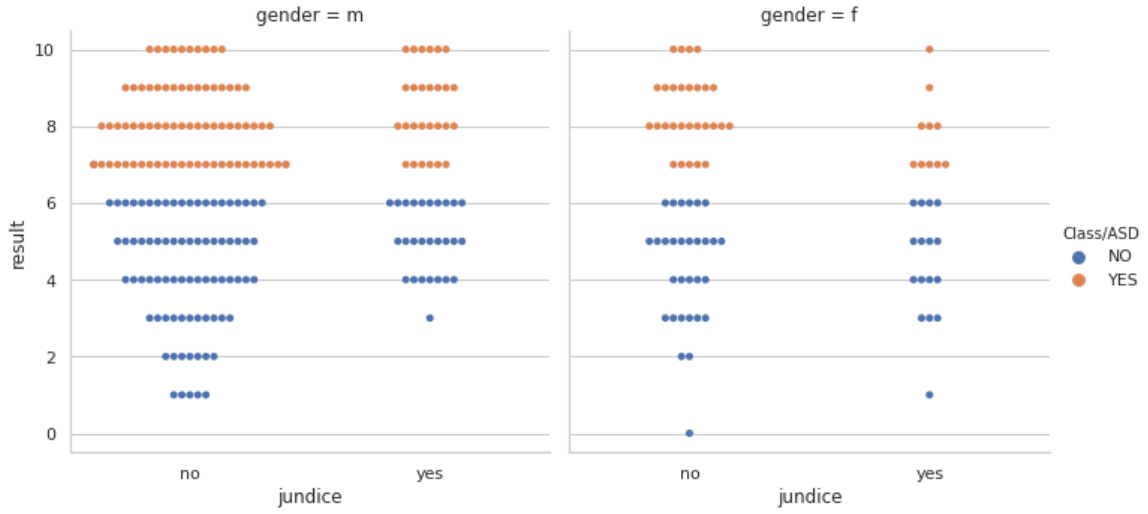
```
[12] sns.factorplot(x="gender", y="result", hue="Class/ASD", col="relation", data=data, kind="box", size=4, aspect=.5, palett
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3669: UserWarning: The `factorplot` function has been renar
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3675: UserWarning: The `size` parameter has been renamed t
  warnings.warn(msg, UserWarning)
```


Next, we present a special form of Factor plot where we display the connection between several attributes from our data set and how they are related with our target class. In this case, we can see when ‘jaundice’ is present at birth, an individual with a higher ‘result’ score will have autism irrespective of their gender.

```
[11] sns.factorplot(x="jundice", y="result", hue="Class/ASD", col="gender", data=data, kind="swarm")
```

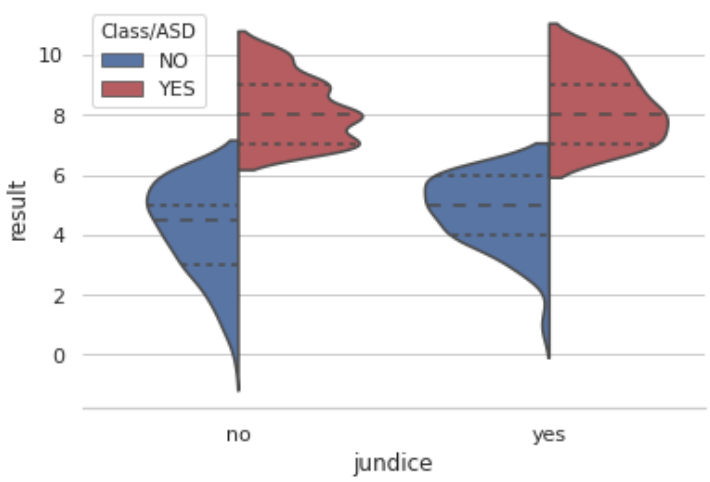
```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3669: UserWarning: The `factorplot` func
  warnings.warn(msg)
<seaborn.axisgrid.FacetGrid at 0x7f7c4f7f8860>
```



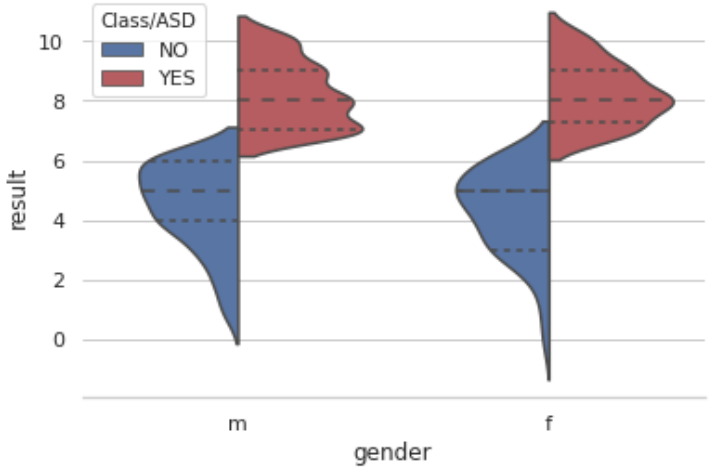

We move on to a series of violin plots to compare how different features contribute to the likelihood of autism. The below figures show a similar relationship between ‘result’ and ‘gender’ versus ‘result’ and ‘jaundice’. In both cases, an individual with a higher ‘result’ score is more likely to have autism, independent of the other feature. The variables ‘jaundice’ or ‘gender’ do not seem to have a lot of influence in deciding the ASD class.

Lastly, we present another variation of a violin plot in Figure 8, where we look at how the distribution of autism varies by relation (Self, Parent, ...), subdivided by whether the patient was born with jaundice (’jaundice’) and the patient’s gender.

```
[9]  # Draw a nested violinplot and split the violins for easier comparison
     sns.violinplot(x="jundice", y="result", hue="Class/ASD", data=data, split=True,
                    inner="quart", palette={'YES': "r", 'NO': "b"})
     sns.despine(left=True)
```



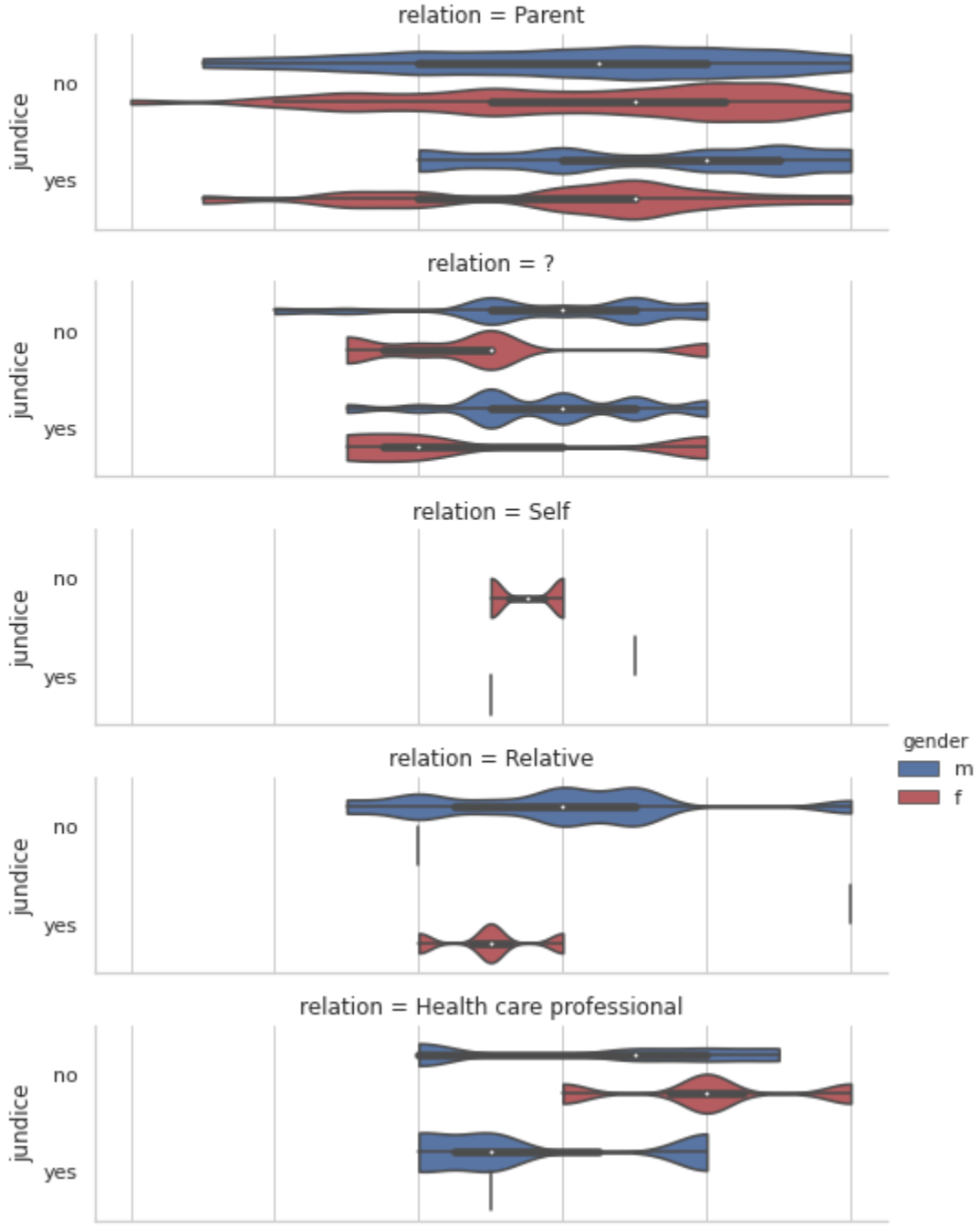

```
     # Draw a nested violinplot and split the violins for easier comparison
     sns.violinplot(x="gender", y="result", hue="Class/ASD", data=data, split=True,
                    inner="quart", palette={'YES': "r", 'NO': "b"})
     sns.despine(left=True)
```




(a) A violine plot showing how the ASD classes is related with the attributes ‘result’ & ‘jundice’.
(b) A violine plot showing how the ASD classes is related with the attributes ‘result’ & ‘gender’.

```
[13] g = sns.factorplot(x="result", y="jundice", hue="gender", row="relation",
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3669: UserWa
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3675: UserWa
  warnings.warn(msg, UserWarning)
```




These dictation exercises are a great way to understand the data anatomy before we decide to apply algorithms which will be most suitable for our goal.

**Algorithms and Techniques:** In this project, I have used Keras to build and train our network. This model is relatively simple and will only use dense (also known as fully connected) layers. This is the most common neural network layer. The network has one hidden layer, use an Adam optimizer, and a categorical cross entropy loss. I did not worry about optimizing parameters such as learning rate, number of neurons in each layer, or activation functions in this project. Below is the model summary:

```
[25] import keras
     # Build a neural network using Keras
     from tensorflow.python.keras.layers import Dense
     from tensorflow.python.keras import Sequential
     from keras.optimizers import Adam

     # Define a function to build the keras model
     def create_model():
         # Create model
         model = Sequential()
         model.add(Dense(8, input_dim=96, kernel_initializer='normal', activation='relu'))
         model.add(Dense(4, kernel_initializer='normal', activation='relu'))
         model.add(Dense(2, activation='sigmoid'))

         # Compile model
         adam = Adam(lr=0.001)
         model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
         return model

     model = create_model()

     print(model.summary())
```

```
Using TensorFlow backend.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 8)                 776
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 36
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 10
=================================================================
Total params: 822
Trainable params: 822
Non-trainable params: 0
_________________________________________________________________
None
```

A **benchmark model**: I'll use the default vanilla model as the benchmark. Hyper parameter tuning my final model will result in significant improvements over this benchmark.

**Data Preprocessing:** This dataset required multiple preprocessing steps. First, I had columns in our DataFrame (attributes) that I don't want to use when training our neural network. I have droped these columns first. Secondly, much of our data is reported using strings; as a result, I have converted our data to categorical labels. During the preprocessing, I have also split the dataset into X and Y datasets, where X has all of the attributes we want to use for prediction and Y has the class labels.

```
[ ]  # drop unwanted columns
     data = data.drop(['id', 'result', 'age_desc'], axis=1)
```

```
[ ]  # create X and Y datasets for training
     x = data.drop(['Class/ASD'], 1)
     y = data['Class/ASD']
```

```
[ ]  # convert the data to categorical values - one-hot-encoded vectors
     X = pd.get_dummies(x)
```

```
[ ]  # print the new categorical column labels
     X.columns.values
```

```
     # convert the class data to categorical values - one-hot-encoded vectors
     Y = pd.get_dummies(y)
```

**Implementation:** I have used Keras to build and train our network. This model is relatively simple and will only use dense (also known as fully connected) layers. This is the most common neural network layer. The network has one hidden layer, uses an Adam optimizer with a learning rate of 0.001, and a categorical cross entropy loss. I have used the activation function as “relu” for the hidden layers and “sigmoid” for the output layer as this is a binary classification problem. Once the model is trained, I have to tested its performance on the testing dataset. The model has never seen this information before; as a result, the testing dataset allows me to determine whether or not the model is able to generalize to information that wasn't used during its training phase. I have used some of the metrics provided by Scikit-learn for this purpose such as classification reports and accuracy score.

```
[ ]  import keras
     # Build a neural network using Keras
     from tensorflow.python.keras.layers import Dense
     from tensorflow.python.keras import Sequential
     from keras.optimizers import Adam

     # Define a function to build the keras model
     def create_model():
         # Create model
         model = Sequential()
         model.add(Dense(8, input_dim=96, kernel_initializer='normal', activation='relu'))
         model.add(Dense(4, kernel_initializer='normal', activation='relu'))
         model.add(Dense(2, activation='sigmoid'))

         # Compile model
         adam = Adam(lr=0.001)
         model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
         return model

     model = create_model()
```

**Results:** I was able to achieve the accuracy on training data as 1.00 whereas on testing data as 96.61%.

```
[27] # generate classification report using predictions for categorical model
     from sklearn.metrics import classification_report, accuracy_score

     predictions = model.predict_classes(X_test)
     predictions
```

```
WARNING:tensorflow:From <ipython-input-27-7b43f079e804>:3: Sequential.pred
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your mod
array([0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1])
```

```
print('Results for Categorical Model')
print(accuracy_score(Y_test[['YES']], predictions))
print(classification_report(Y_test[['YES']], predictions))
```

```
Results for Categorical Model
0.9661016949152542
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        24
           1       0.97      0.97      0.97        35

    accuracy                           0.97        59
   macro avg       0.96      0.96      0.96        59
weighted avg       0.97      0.97      0.97        59
```

**Improvement:** The accuracy can on the testing data can be further improved by tuning the hyper parameters, learning rate and activation functions. Also various other algorithms can be tried as RMSProd, Gradient Descent, BFGS or L-BFGS.