A Machine Learning Approach to Cardiotocography Interpretation

Milestone 4: Finalizing Your Results

Holly Figueroa
KJ MoChroi
DSC 630 Winter 2022
Bellevue University

Milestone 2: Preliminary Analysis

A Machine Learning Approach to Cardiotocography Interpretation

Despite modern advances, childbirth remains a risky medical endeavor both for birthing people and the unborn. There exists no replacement for the expertise and direct involvement of a medical professional to manage care. Analytics, however, can offer significant aid in ways it could not before through advanced modeling. Cardiotocography (CTG) is widely used over the course of pregnancy and during labor (Fetal Heart Monitoring, 2019). CTG allows monitoring of fetal heart rate and uterine contractions to prevent fetal hypoxia, or oxygen loss in body tissues (Boudet et al., 2020). Creating effective modeling for CTG exam data can provide additional support to those dedicated preventing harm and loss of life. The goal of this project will be to build an effective model to classify CTG data using supervised machine learning.

Data Selection

The dataset chosen for model training contains 2126 instances of heart-health-related features taken from cardiotocogram exams of fetuses. These were then classified by three expert obstetricians into 3 classes: 1 – Normal, 2 – Suspect, or 3- Pathological, making the target a multi-class categorical feature. There are a total of 22 features, both numerical and categorical. We found this dataset on Kaggle, but it was originally published in the Journal of Maternal-Fetal & Neonatal Medicine in the year 2000. This dataset should be useful for creating a model that can predict suspect and pathological health outcomes, with the assumption that some of the features provided will be correlated with fetal heart health.

Model Selection & Evaluation

We plan to implement two different models for this project, both of which will perform classification on the fetal heart health target feature. The first model will be a neural network. We think that this dataset provides a good opportunity to train a neural network because of the substantial number of instances in the data. Neural networks are also considered some of the most accurate models, and they are ideal in a medical setting like this where accuracy is important. We plan to use the percentage of accuracy to score and evaluate this model. In this context where we are discussing medical diagnostics, a false negative has more detrimental outcomes to the patient's life than a false positive. Thus, we will review the confusion matrices in addition to the accuracy percentage, so that we can calculate rates for false positives and false negatives.

The second model we plan to use is Naïve Bayes classification model. Although the dataset contains over 2,000 instances, it is possible that this amount of data is not sufficient to train a competent neural network. The Naïve Bayes model is recommended as an alternative in that circumstance. It also has a reputation as an effective classifier and has been proven useful in medical diagnostics (Simfukwe et al., 2015). Since we are still doing classification, we will be using accuracy and confusion matrices to evaluate the results of our model. We can compare the accuracy score of this model to the neural network model in this proposal.

To ensure quality and accuracy, we plan to implement a variety of techniques to improve both models. We will need to clean the data, do some feature scaling, and feature selection. If there are any categorical features that are relevant to the model, we can create dummy variables with them before training the model. We plan to use a grid search to perform hyperparameter tuning on our models as well. Since the target feature has imbalanced classes, we will need to perform oversampling on the underrepresented classes.

Risks and Ethical Implications

Machine learning and artificial intelligence has already earned space in healthcare with the increased use of AI diagnostic systems. Some common examples involve the interpretation of x-rays, MRI scans, CT scans, and other image-related tools of measurement. While machine learning healthcare applications (ML-HCAs) and AI recommendation systems can offer powerful insights for the medical field, limits must be acknowledged. The most fundamental limit being correlation is not causation, therefore their role must be additive to care, not a replacement for expertise. A common hope is that the addition of ML-HCAs and AI diagnostics will circumvent bias within the healthcare system, however, these tools are just as suspectable (Naik, 2022). The data used to train a diagnostic model is critical to how accurately these systems perform for diverse populations. It is unknown how these concerns apply to CTG exams of the unborn. For example, we do not know if CTG exams, in practice, predict or protect all groups equally. We also do not know at this time if the anomalies associated with fetal heart issues present the same across groups. For this project, however, we will assume the data is sound towards its purpose.

Contingency Proposal

In the case that our original proposal is unsuccessful, the contingency plan will be to use an alternative dataset to train a regression model. We found a dataset that is intended to train models to predict the insurance premium price of an individual given certain health history data points about them. This dataset has 10 predicting features about the health history of an individual and 986 instances with no missing values. Since our target is continuous and we want to include multiple features, we will use a multiple regression model.

Milestone 3

Will I be able to answer the questions I want to answer with the data I have?

Our preliminary analysis suggests no reason our data should not perform suitably for modeling with some adjustments for scaling and normalization. No missing data was found. Histograms and descriptive statistics were used to conduct univariate analysis on all variables. Multivariate analysis was conducted using correlations matrices and box plots to illustrate variable distributions across our 3 target classes for fetal health outcome: Normal(1), Suspect(2), and Pathological(3).

Our preliminary analysis with correlations showed that variation in many of our features are associated with variations in our target classes. The relationships are not necessarily strong, but combined, they will likely offer appropriate training for our predictive models. The strongest relationships to our target classes were related to CTG features involving deceleration at .50 Tau and abnormal variability, with those Tau's ranging from .25 to .43

What visualizations are especially useful for explaining my data?

Histograms, while simple, were in many ways most effective in conveying data issues to consider. Knowing data is skewed, differently scaled, imbalanced, or lacking variation in measures, lends a lot for our preprocessing needs as mentioned above. Histograms related to heart decelerations per second were found to be highly imbalanced. But upon further inspection using value counts, the rare values represented cases with the most serious health outcome, 'pathological'. Box were effective in allowing us to understand how our features distribute and relate to our target classes. Short-term abnormal variability appeared to have a positive linear relationship across our target classes as they ranged from normal health to pathological.

Do I need to adjust the data and/or driving questions?

The analysis shows that some steps, such as scaling, and normalization for some variables will have to be taken prior to being used as input for our models. Strong correlations between variables were also shown between different central measures related to ETC histograms (mean, median, and mode), which led to the dropping of variables for median and mode. We also know that our target data will require us to use sampling techniques to balance our target classes when model training. So, while changes will have to made, the data itself and questions remain appropriate.

Do I need to adjust my model/evaluation choices?

At this time, our model choices appear reasonable. We have chosen both a neural network classifier and a naive Bayes classifier for modeling and comparison. It is possible that the independent treatment of features within a naive Bayes may not be as well suited. The features may share probabilistic qualities with each other given the specialized topic. Box plots also illustrate what appear to be nuanced difference in feature distributions across target classes. While we do not have the domain expertise to objectively know, our current model choices can be compared to offer some insights. The confusions matrices we have chosen to evaluate results will help illustrate how well our model discerns between classes.

Are my original expectations still reasonable?

Given the steps we can take to prepare data for modeling, our plans for modeling seem reasonable. We will have a neural network classifier and a naïve Bayes classifier planned to compare. Our preliminary analysis shows no indication an effective model cannot result from this dataset. The target classes may be imbalanced, but we can utilize over sampling techniques to give our model more proportionate data to work with. The data has no missing values, and there is no reason to believe any of the observations were made in error. We also already know that many of our features have some established relevance to predicting fetal outcomes as they represent output features from existing diagnostic techniques.

Data Preparation

All the data preparation done for these models is described in detail in Milestone 3. In summary, data preparation consisted of removing highly correlated features with a threshold of correlation above 0.9. By this criterion there were two features that were removed.

In [1]:

```
# Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import ComplementNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_validate



from imblearn.over_sampling import RandomOverSampler
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

In [2]:

```
#    Title: get_tpr_fnr_fpr_tnr()
#    Author: Md Abdul Bari
#    Date: Oct 22, 2020
#    Availability: https://stackoverflow.com/questions/45053238/how-to-get-all-confusio

def get_tpr_fnr_fpr_tnr(cm):
    """
    This function returns class-wise TPR, FNR, FPR & TNR
    [[cm]]: a 2-D array of a multiclass confusion matrix
            where horizontal axes represent actual classes
            and vertical axes represent predicted classes
    {output}: a dictionary of class-wise accuracy parameters
    """
    dict_metric = dict()
    n = len(cm[0])
    row_sums = cm.sum(axis=1)
    col_sums = cm.sum(axis=0)
    array_sum = sum(sum(cm))
    #initialize a blank nested dictionary
    for i in range(1, n+1):
        keys = str(i)
        dict_metric[keys] = {"TPR":0, "FNR":0, "FPR":0, "TNR":0}
    # calculate and store class-wise TPR, FNR, FPR, TNR
    for i in range(n):
        for j in range(n):
            if i == j:
                keys = str(i+1)
                tp = cm[i, j]
                fn = row_sums[i] - cm[i, j]
                dict_metric[keys]["TPR"] = tp / (tp + fn)
                dict_metric[keys]["FNR"] = fn / (tp + fn)
                fp = col_sums[i] - cm[i, j]
                tn = array_sum - tp - fn - fp
                dict_metric[keys]["FPR"] = fp / (fp + tn)
                dict_metric[keys]["TNR"] = tn / (fp + tn)
    return dict_metric
```

In [3]:

```
df = pd.read_csv('fetal_health.csv')
df.head()
```

Out[3]:

	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	severe_decelerations
0	120.0	0.000	0.0	0.000	0.000	0.0
1	132.0	0.006	0.0	0.006	0.003	0.0
2	133.0	0.003	0.0	0.008	0.003	0.0
3	134.0	0.003	0.0	0.008	0.003	0.0
4	132.0	0.007	0.0	0.008	0.000	0.0

5 rows × 22 columns

In [4]:

```
# Drop highly correlated variables
# 0.9 Threshold as described in Milestone 3

# Histogram mode and median highly correlate with hisogram mean
df.drop(['histogram_median', 'histogram_mode', ], axis=1, inplace=True)
```

In [5]:

```
# seperate target feature
X = df.drop(['fetal_health'], axis=1)
y = df['fetal_health']
```

In [6]:

```
# train test split data using stratify to preserve the class ratios
train_X, test_X, train_y, test_y = train_test_split(X,y, test_size = 0.2, stratify = y,
# Check shape
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(1700, 19) (1700,) (426, 19) (426,)
```

Null Model for Comparison

Below we will calculate the accuracy of a model that simply always chooses the most likely outcome. In this case the model will predict that all fetal heart health outcomes are healthy.

In [7]:
```
# check class distribution in test set
test_y.value_counts()
```

Out[7]:
```
1.0    332
2.0     59
3.0     35
Name: fetal_health, dtype: int64
```

In [8]:
```
# check null accuracy score
null_accuracy = (497/(497+94+47))
print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.7790
```

In [9]:
```
# So we need to score above 78% to be a good model
```

Complement Naive Bayes

Preparation

For the Naïve Bayes classifier, we chose to use the Complement Naïve Bayes due to its ability to handle imbalanced classes in the target feature. We used a train-test-split ratio of 80/20 because it yielded the highest accuracy of any ratio we tested, without overfitting the model too much. We then applied a minmax scalar because the model only accepts positive values as input. After that we did a grid search to perform hyperparameter tuning on the only hyperparameter in a complement naïve Bayes model, using accuracy as our scoring metric.

In [10]:
```
# Scaling
scaler = MinMaxScaler()
# I had to use this particular scalar because of negative numbers error
x_train = scaler.fit_transform(train_X)
x_test = scaler.transform(test_X)
```

Hyperparameter Tuning for Model Training

In [11]:
```
# gridsearch values
grid_vals = {"alpha": [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
                        1, 2, 3, 4, 5, 6, 7, 8, 9,
                       10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
                       20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
                       30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
                       40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
                       50, 51, 52, 53, 54, 55, 56, 57, 48, 59,
                       60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
                       70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
                       80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
                       90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]}
```

In [12]:
```
# gridsearch model
grid_lr = GridSearchCV(estimator=ComplementNB(), param_grid=grid_vals, scoring='accurac
                       cv=6, refit=True, return_train_score=True)
```

In [13]:
```
#Training and Prediction
tuned_classifier = grid_lr.fit(x_train, train_y)
tuned_preds = grid_lr.best_estimator_.predict(x_test)
```

In [14]:
```
# Print the tuned parameters and score
print("Tuned Gaussian NB Parameter: {}".format(grid_lr.best_params_))
```

```
Tuned Gaussian NB Parameter: {'alpha': 42}
```

Evaluation

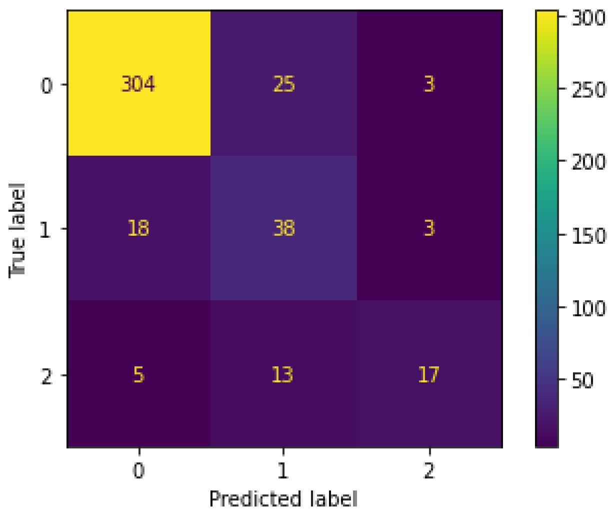
In [15]:
```
# cross validate
scores3 = cross_validate(tuned_classifier, x_train, train_y, return_train_score=True)
scores_df3 = pd.DataFrame(scores3)
```

In [16]:
```
scores_df3.mean()
# Looks like we could be over fitting but the model is performing better than null
```

Out[16]:
```
fit_time       1.018108
score_time     0.000402
test_score     0.840588
train_score    0.839559
dtype: float64
```

In [17]:
```
# use display matrix function to heatmap prediction/true cases
cm = confusion_matrix(test_y, tuned_preds)
disp = ConfusionMatrixDisplay(cm)
disp.plot()

plt.show()
```



In [18]:

```
# Print the Confusion Matrix

conf_mat = pd.DataFrame(get_tpr_fnr_fpr_tnr(cm)).transpose()
conf_mat
```

Out[18]:

	TPR	FNR	FPR	TNR
1	0.915663	0.084337	0.244681	0.755319
2	0.644068	0.355932	0.103542	0.896458
3	0.485714	0.514286	0.015345	0.984655

Multilayer Perceptron NN Classifier

Preparation

For our neural network model, a multilayer perceptive classifier was chosen for our project. Given the strong imbalances between classes, data was train-test-split in a stratified manner. This offered assurance that classes with small counts were not further reduced during the split. A train-test-split ratio of 80/20 was chosen to offer our model a wider range of training cases to learn from. To further accommodate the imbalance between class counts, oversampling was also conducted to increase class counts to match the larger class count, of 1323. This method was chosen instead of under-sampling to better complement this model type which is characterized to benefit from larger training sets. Two scalers were tested, and a standard scaler proved to provide slightly higher accuracy.

In [19]:

```
# Random Oversampling
over_sampler = RandomOverSampler(sampling_strategy = 'auto', random_state = 0)
ros_X, ros_y = over_sampler.fit_resample(train_X,train_y)
```

In [20]:

```
# Check counts/oversampling results
ros_y.value_counts()
```

Out[20]:

```
2.0    1323
1.0    1323
3.0    1323
Name: fetal_health, dtype: int64
```

In [21]:

```
# Encode classes so they are 0-2 instead of 1-3
# Now training y and test_y will be label encoded and oversampled
enc = preprocessing.LabelEncoder()
ros_y = enc.fit_transform(ros_y)
test_y2 = enc.transform(test_y)
```

In [22]:

```
# Create scaler
scaler = StandardScaler()
# Create Model Instance for gridsearching parameters
mlp = MLPClassifier()
```

```
# Create pipeline
pipe = Pipeline(steps=[('scaler', scaler ), ('mlpc', mlp)])
```

Hyperparameter Tuning

In [23]:
```
#mlp.get_params()
```

In [24]:
```
# # Define gridspace
# param_grid= [{'mlpc__activation': ['relu','tanh'],
#              'mlpc__alpha': [0.0025, 0.05, 0.5 ],
#              'mlpc__hidden_layer_sizes': [(12,6)],
#              'mlpc__max_iter': [2000],
#              'mlpc__solver': ['adam','lbfgs'],
#              'mlpc__batch_size': [20,30,50],
#              'mlpc__learning_rate':['invscaling','adaptive']
#              }]
```

In [25]:
```
# # Run a new gridsearch with new search_space
# grid_mlp = GridSearchCV(pipe, param_grid, n_jobs= -1, cv = 2, verbose = 0)
```

In [26]:
```
# # Fit grid search
# grid_mlp.fit(ros_X,ros_y)
```

In [27]:
```
# grid_mlp.best_params_
```

In [28]:
```
# print('Best score: ', grid_mlp.best_score_)
# print('Test Accuracy: ', grid_mlp.score(test_X, test_y))
```

In [29]:
```
# Create MLP model with chosen params
best_mlp = MLPClassifier(
                        activation= 'tanh',
                        alpha = 0.001,
                        hidden_layer_sizes = (12,6),
                        max_iter = 2000,
                        solver = 'lbfgs',
                        random_state = 12,
                        learning_rate = 'adaptive'
                        )
```

In [30]:
```
# Update pipeline
pipe = Pipeline(steps=[('scaler', scaler ), ('mlpc', best_mlp)])

# Fit to features
pipe.fit(ros_X, ros_y)

# Get predictions
pred_y = pipe.predict(test_X)
```

Evaluation

In [33]:

```
# Get accuracy of predictions
accuracy_score(test_y2, pred_y)
```

Out[33]:

```
0.9107981220657277
```

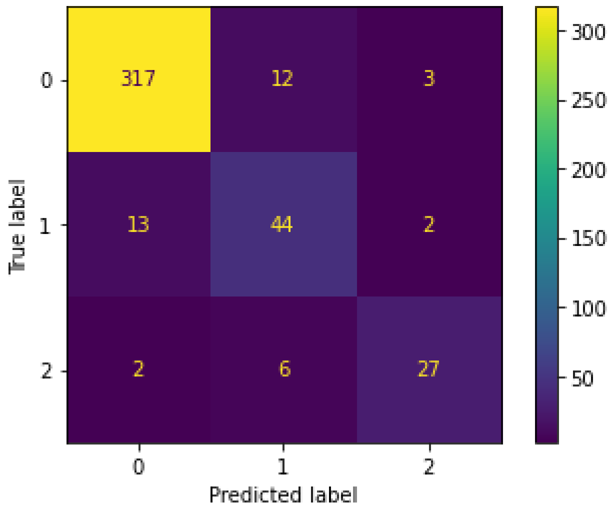
In [34]:

```
# Create confusion matrix
cm = confusion_matrix(test_y2, pred_y)
```

In [35]:

```
# use display matrix function to heatmap prediction/true cases
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()

plt.show()
```



In [36]:

```
rates_df = pd.DataFrame(get_tpr_fnr_fpr_tnr(cm)).transpose()
rates_df
```

Out[36]:

	TPR	FNR	FPR	TNR
1	0.954819	0.045181	0.159574	0.840426
2	0.745763	0.254237	0.049046	0.950954
3	0.771429	0.228571	0.012788	0.987212

Results

In evaluation of the complement naïve Bayes model, we primarily used accuracy but also looked more closely at false positive rates, false negative rates, true positive rates, and true negative rates. The k-fold cross validation of this model has a train score of 0.84 and a test score of 0.84. This could imply some overfitting is happening within this model. Furthermore, when we look more closely at the true positive rates for each of the categorical outcomes, we do have some serious concerns. It appears that this model is very accurate at predicting healthy outcomes, with a true positive rate of 91%, but much worse at correctly predicting other fetal health outcomes, with true positive rates of less than 65% for each of the other outcomes.

For our MLP classifier, a grid search was conducted to find ideal model hyperparameters. Cross validation however, revealed a lower performance when predicting test data, indicating overfitting. The hyperparameters were tuned manually with cross validation to reach a higher accuracy in predicting outcomes for our test data. The resulting classifier model calculated outcomes with a .91 accuracy score. Outcomes for 'normal', 'suspect', and 'pathological' were returned with True Positive Rates of 95%, 74%, and 80%, respectively. Of particular concern, given the context of this project, were False Negative Rates for non-normal outcomes. Results returned a FNR of 25% for 'suspect' and 20% for 'pathological' predictions. While this lends confidence for predicting 'normal' outcomes, overall, the results show the model is not dependable for predicting riskier outcomes.

Conclusion and Ethical Implications

It is worth noting that both of our models performed much better than the null model with regards to accuracy and in that way, they can be considered successful models. When comparing both model results it appears that the MLP neural network model performed better with an overall accuracy of 91% compared to our Complement Naïve Bayes accuracy at 84%. Both models, however, under-performed when correctly classifying 'suspect' and 'pathological' outcomes with false negative rates ranging from 20%-51% across both models. Since the purpose of these models is to identify unhealthy fetal heart health outcomes, we feel that these models are not accurate enough to be deployed for diagnostic purposes. It is possible that the imbalance between classes and the nuanced differences between class cases are bit ill-suited for training these model types. It would be interesting to see how results compared with a tree-based model. Although the purpose of this diagnostic tool is to assist physicians in the accurate assessment of fetal heart hearth, if such a tool were to eventually replace physicians, that could result in a lower quality of care for patients.

References

- Boudet, S., Houzé l'Aulnoit, A., Demailly, R., Delgranche, A., Peyrodie, L., Beuscart, R., & Houzé de l'Aulnoit, D. (2020). A fetal heart rate morphological analysis toolbox for MATLAB. SoftwareX, 11, 100428. https://doi.org/10.1016/j.softx.2020.100428

- Fetal Heart Monitoring. (2019, August 14). Johns Hopkins Medicine. https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/fetal-heart-monitoring

- Naik, N. (2022, March 14). Legal and Ethical Consideration in Artificial Intelligence in Healthcare: Who Takes Responsibility? Frontiers. https://www.frontiersin.org/articles/10.3389/fsurg.2022.862322/full

- Simfukwe, M., Kunda, D., & Chembe, C. (2015). Comparing Naive Bayes Method and Artificial Neural Network for Semen Quality Categorization. International Journal of Innovative Science, Engineering & Technology, 2(7) https://ijiset.com/vol2/v2s7/IJISET_V2_I6_90.pdf

- Md Abdul Bari (Oct 22, 2020) get_tpr_fnr_fpr_tnr() [Python Function]. https://stackoverflow.com/questions/45053238/how-to-get-all-confusion-matrix-terminologies-tpr-fpr-tnr-fnr-for-a-multi-c.