Machine Learning: Fetus Classification

Group 1
Ben Stork
Julie Pyle
Segun Olorunfemi

Important Questions

We had three questions we wanted to try and answer.

1. Can we use machine learning to predict healthcare outcomes?
2. How many input columns will be needed to maintain a minimum of 90% accuracy?
3. Can we increase the accuracy of the model by tuning the parameters?

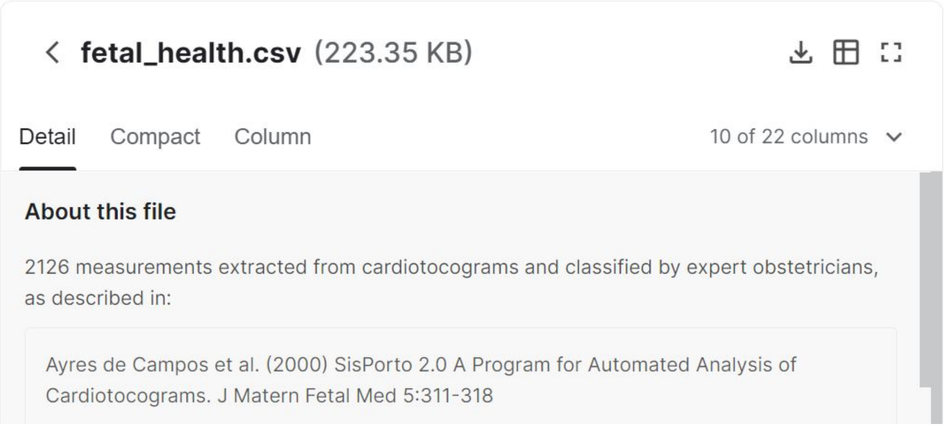
The group was originally interested in exploring machine learning in the medical field. We were able to find the data set on Kaggle that met some desired requirements including:

1. More than 1,000 data points
2. Numerical Data
3. Final classification into 3 or less categories

Our Data Sources

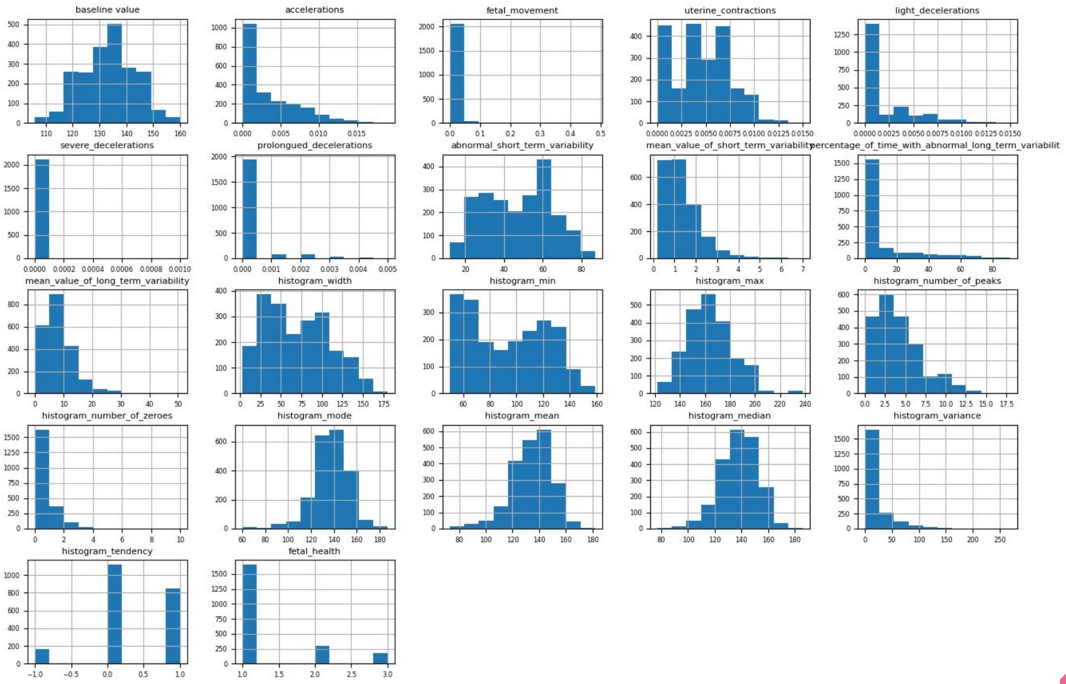
The Following Data Source Was Utilized:

Fetal Health Classification- Ayres de Campos et al. (2000) SisPorto 2.0 A Program for Automated Analysis of Cardiotocograms. J Matern Fetal Med 5:311-318
https://www.kaggle.com/andrewmvd/fetal-health-classification



Input Histograms:

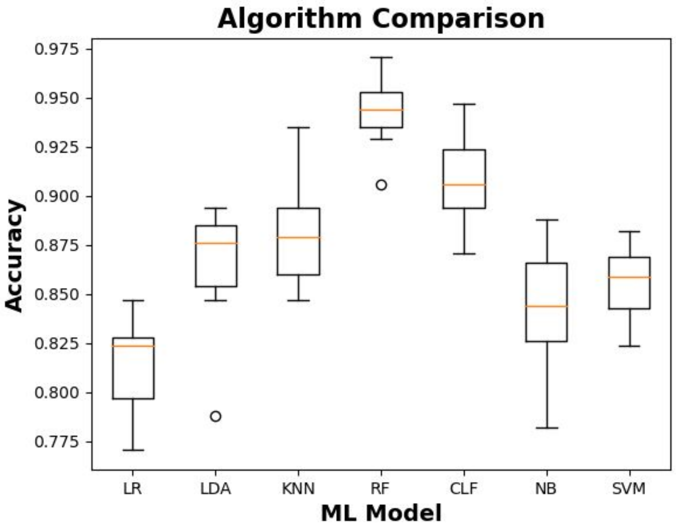
Below are the histograms of the input data from the dataset.



Checking ML Algorithms

The following code was utilized to check which machine learning algorithm would give the highest accuracy to determine what would be used going forward.

```python
# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('RF', RandomForestClassifier()))
models.append(('CLF', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
```



Parameter Tuning/Final Accuracy

```python
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 1200, num = 11)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)
```

Code to set up the testing grid

```python
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf1 = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available
cores
rf_random = RandomizedSearchCV(estimator = rf1, param_distributions =
random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs
= -1)
# Fit the random search model
rf_random.fit(X_train5, y_train)
```

Code to Test for best Parameters

Outcomes of Hypertuning

```
M↓
# Use hypertuning parameters to train random forest classifier (ran
previously and these were the numbers that were obtained)
final_model = RandomForestClassifier(n_estimators=200,
min_samples_split=2, min_samples_leaf=1, max_features='sqrt',
max_depth=20, bootstrap=True)
final_model = final_model.fit(X_train5, y_train)
final_model.score(X_test5, y_test)

0.9295774647887324
```

93% Accuracy was Achieved with Tuned Model for Test Data

```
M↓
# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test5, y_test)
print(result)

0.9295774647887324
```

Confirmation of Loaded Model Accuracy

Initial thoughts on Visualization

public.fetal_upload/postgres/postgres@CWRU-Project

Messages | Data Output | Explain | Query Editor | Query History

	index bigint	baseline value double precision	accelerations double precision	uterine_contractions double precision	prolongued_decelerations double precision	abnormal_short_term_variabilit double precision	percentage_of_time_with_abnormal_long double precision	fetal_health double precision
1	0	120	0	0	0	73	43	
2	1	132	0.006	0.006	0	17	0	
3	2	133	0.003	0.008	0	16	0	
4	3	134	0.003	0.008	0	16	0	
5	4	132	0.006999999999999999	0.008	0	16	0	
6	5	134	0.001	0.01	0.002	26	0	
7	6	134	0.001	0.013000000000000001	0.003	29	0	
8	7	122	0	0	0	83	6	
9	8	122	0	0.002	0	84	5	
10	9	122	0	0.003	0	86	6	
11	10	151	0	0.001	0	64	9	
12	11	150	0	0.001	0	64	8	
13	12	131	0.005	0.008	0	28	0	
14	13	131	0.009000000000000001	0.006	0	28	0	
15	14	130	0.006	0.004	0.001	21	0	

The data on Postgres with AWS server

```
[18]
alchemyEngine   = create_engine('postgresql+psycopg2://postgres:' + password + '@final-project.cbgqvzvry5u3.us-east-2.rds.amazonaws.com/postgres', pool_recycle=3600)

# Connect to PostgreSQL server
postgreSQLConnection = alchemyEngine.connect()
postgreSQLTable = "fetal_upload"

try:

    frame = fetal_upload.to_sql(postgreSQLTable, postgreSQLConnection, if_exists='fail')

except ValueError as vx:

    print(vx)

except Exception as ex:

    print(ex)

else:

    print("PostgreSQL Table %s has been created successfully."%postgreSQLTable)

finally:

    postgreSQLConnection.close()

PostgreSQL Table fetal_upload has been created successfully.
```

The code uploading on Postgres AWS server

From ML to Visualization - Flask App

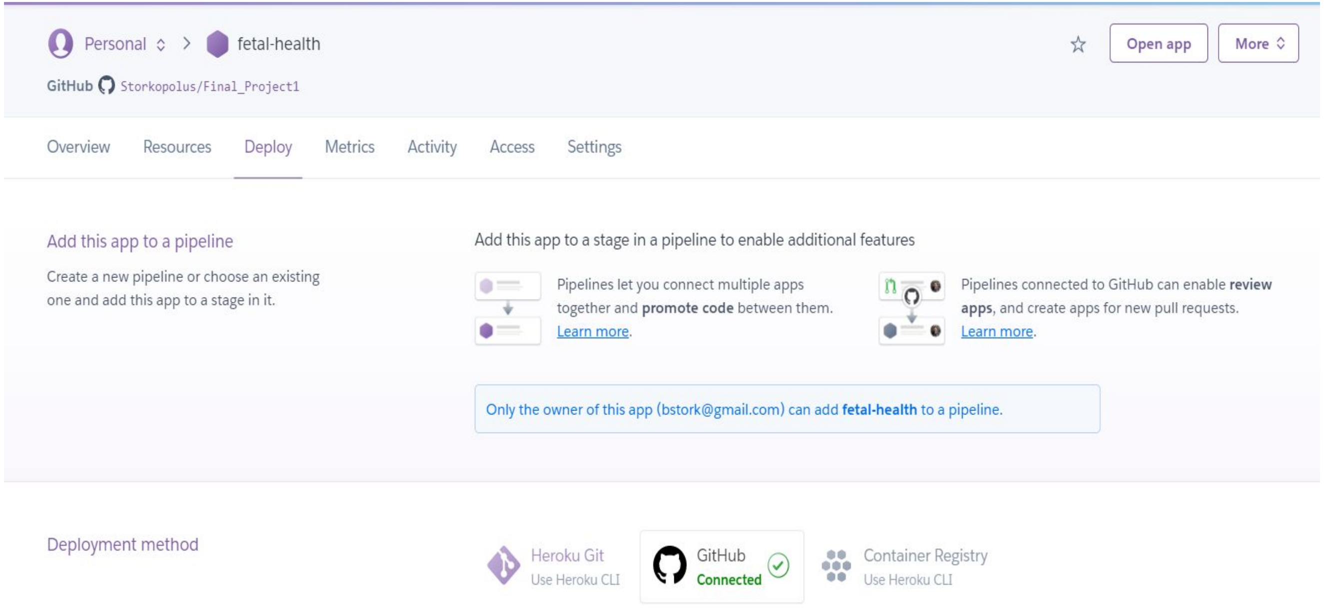
```
# clf = load('finalized_model.sav')
@app.route('/predict', methods=['POST'])
def predict():
    # try out for code on Heroku
    clf = pickle.load(open('finalized_model.sav', 'rb'))
    print('clf')
    int_features = [float(x) for x in request.form.values()]
    query_df = [np.array(int_features)]
    print(query_df)
    # query = pd.get_dummies(query_df)

    # for col in model_columns:
    #     if col not in query.columns:
    #         query[col] = 0

    prediction = clf.predict(query_df)
    print(prediction)
    prediction_dictionary = {
        1: "The health of the fetus is likely normal.",
        2: "The health of the fetus is suspicious for possible pathology.",
        3: "The health of the fetus is likely pathological."
    }
    if int(prediction) in prediction_dictionary.keys():
        prediction_string=prediction_dictionary[int(prediction)]
    return render_template('fetal_health_predictor.html', prediction = prediction_string)
```

Flask app was used to build user interaction with the ML done. Predict route takes information from user and presents result.

Deployment of app on the Internet



Heroku was used in deploying app noting

- its ease of use,
- ability to scale application
- suitable for web and python based applications

6 User Inputs for Machine Learning

Baseline Fetal Heart Rate: the heart rate during a 10 minute segment rounded to the nearest 5 beats per minute. Normal baseline fetal heart rate is 110-160 beats per minute.

Accelerations: short term rises in the baseline fetal heart rate of at least 15 beats per minute and lasting at least 15 seconds. These are normal and healthy, and indicate the fetus has an adequate supply of oxygen.

Uterine Contractions: tightening of the muscles of the uterus. During the 2nd and 3rd trimester, occasional contractions called Braxton-Hicks are normal and may last 30 seconds to 2 minutes. During labor, contractions last 30-70 seconds and become more frequent as the baby is ready to be delivered.

6 User Inputs for Machine Learning

Decelerations: decelerations are temporary drops in the baseline fetal heart rate. A prolonged deceleration is a drop below baseline by 15 bpm for longer than 2 minutes but less than 10 minutes. Prolonged deceleration indicate the fetus isn't getting enough oxygen.

Short Term Variability: short term variability is the beat to beat variation in fetal heart rate. Normal variability is 6-25 beats per minute. Variability is normal after 32 weeks. Decreased variability may indicate lack of oxygen or a congenital heart anomaly.

Long Term Variability: long term variability is the fluctuation in fetal heart rate in one minute.

Output Possibilities

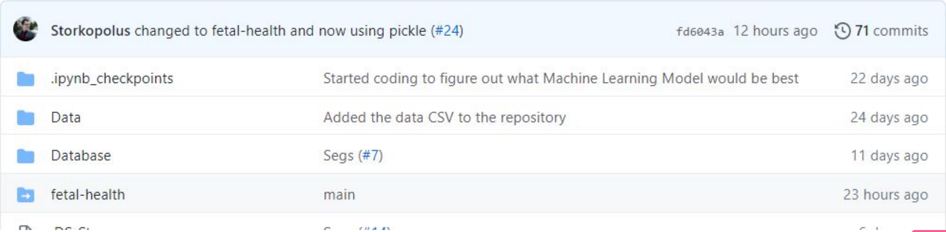
Machine Learning outputs a 1, 2 or 3 as the result

Created a dictionary in the Flask app to output a String related to the numerical output

```
prediction_dictionary = {
        1: "The health of the fetus is likely normal.",
        2: "The health of the fetus is suspicious for possible pathology.",
        3: "The health of the fetus is likely pathological."
}
```

Conclusions/Learning lessons

- To deploy to Heroku, need to make sure have all proper requirements
- Learned and experimented with Heroku--loading from github vs. terminal/bash
- Needed to confirm that machine learning model was in the correct folder location.
    - There were issues with Heroku finding our finalized_model.sav
- When deploying Heroku app (from GitHub), make sure that the person hosting the repository is the one that launches the application. Otherwise there could be additional folders created inside the repository.



Show Website