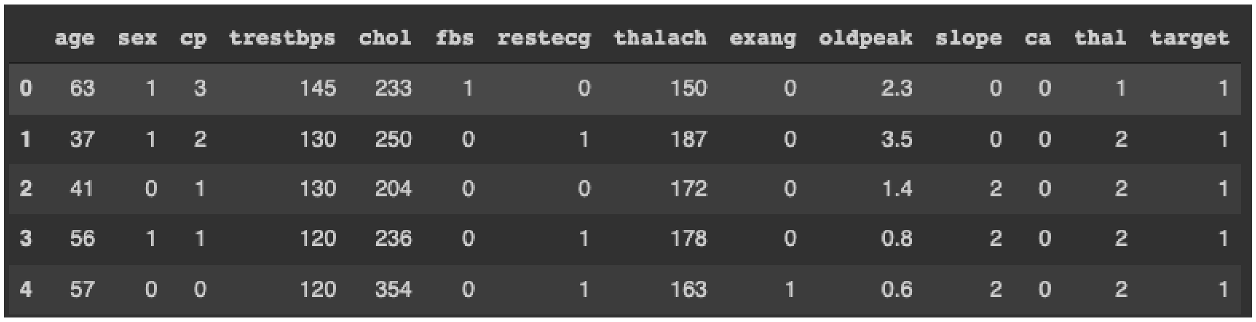
CLASSIFICATION OF HEART DISEASE USING VARIOUS METHODS

for
Paul Anderson
Computer Science Professor
California Polytechnic State University
San Luis Obispo, California

by
Keerat Grewal, Phaniraj Aenugula
CSC 466 Students

Introduction

Heart disease is the leading killer taking around 700,000 lives each year. One of the best ways to fight this disease is to detect early presence and tackle the disease as soon as possible. We wanted to see if it is possible to classify if an individual has heart disease based on certain features such as sex, cholestorol levels, thalassiamia presence, etc. The dataset contains a ‘target’ column which has one of two values: 0 or 1. ‘0’ means there is no presence of heart disease in the individual, while ‘1’ means there is presence of heart disease.



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Heart Disease Dataset

Methods

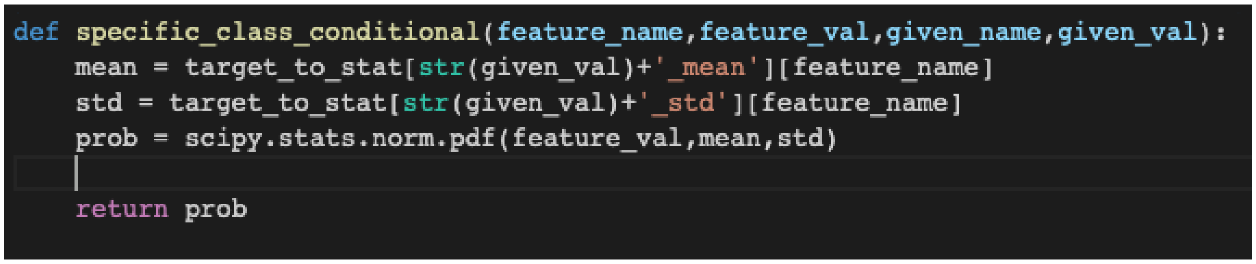
Gaussian Bayesian Classifier

The Gaussian Bayesian Classifier assumes that the data is normally distributed, therefore we can calculate the probabilities directly using basic statistical calculations for normalized data. The calculation to get the probability of target = 1 or target = 0 given the values of all the other features in the dataset is shown in the figure below. The probability needs to be split up into parts for each feature as outlined in Bayes rule. We also need the prior probabilities as part of our calculation of the probabilities.

$$\Pr(target = 1|trestbps, chol, etc..) = \frac{\Pr(trestbps|target{=}1){*}\Pr(chol|target{=}1), etc.{*}\ \Pr(target{=}1)}{\Pr(trestbps|target{=}1){*}\Pr(chol|target{=}1), etc.{*}\ \Pr(target{=}1){+}\Pr(trestbps|\neg target{=}1){*}\Pr(chol|\neg target{=}1), etc.{*}Pr(\neg target{=}1)}$$

Probability Calculation

Once the probability is calculated for each target value, the prediction is made based on which probability is greater than fifty percent.



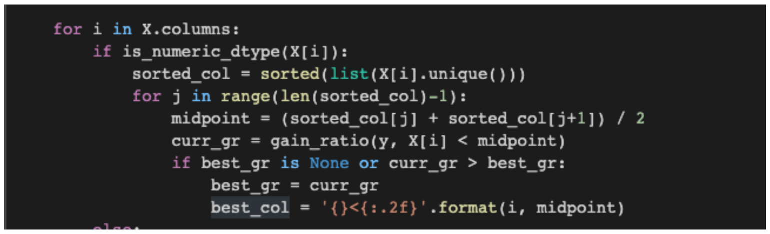
```python
def specific_class_conditional(feature_name,feature_val,given_name,given_val):
    mean = target_to_stat[str(given_val)+'_mean'][feature_name]
    std = target_to_stat[str(given_val)+'_std'][feature_name]
    prob = scipy.stats.norm.pdf(feature_val,mean,std)

    return prob
```

Function for calculating specific class conditional

As shown in the figure above, for each feature we need to calculate the conditional probability. This is done by using a built-in library called scipy.stats to calculate the pdf value, which will be the probability. The pdf value needs the standard deviation and mean of the current feature we are looking at.

Decision Tree

We used a c45 decision tree to make the predictions. We create bounds for each of the features in the data by taking the average of each entry on each feature.



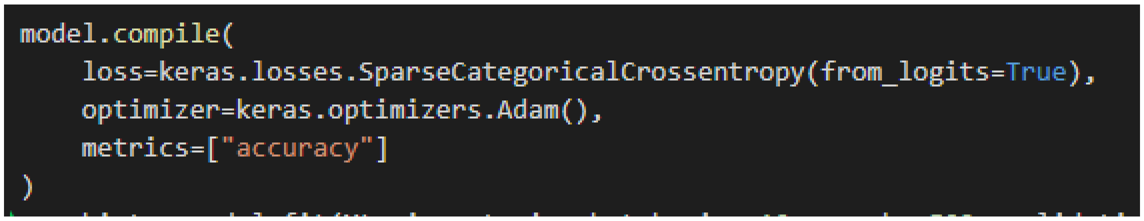
```python
for i in X.columns:
    if is_numeric_dtype(X[i]):
        sorted_col = sorted(list(X[i].unique()))
        for j in range(len(sorted_col)-1):
            midpoint = (sorted_col[j] + sorted_col[j+1]) / 2
            curr_gr = gain_ratio(y, X[i] < midpoint)
            if best_gr is None or curr_gr > best_gr:
                best_gr = curr_gr
                best_col = '{}<{:.2f}'.format(i, midpoint)
```

The code in this image creates these bounds while deciding which column has the best gain ratio so that we take that path in the tree. Gain ratio is calculated by taking the information gain and dividing that by entropy. After the tree is created, we can now generate the rules by traversing each path of the tree. This will give us the basis for our predictions when we encounter a new item we desire to predict.

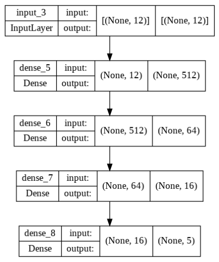
Neural Network

Our neural network for classification of the heart disease traits required both some guessing and testing as well as some preprocessing work. The neural network we had implemented entirely from scratch uses standard gradient descent to periodically update weights and improve accuracy. This required preprocessing of the data along a normal curve.

Our tensorflow implementation involved testing out the various features and usages of the neural network models. They had quite a few implementations of different layers, optimizers, activations functions and even loss functions. We implemented the following model using the below code to compile and run our model.



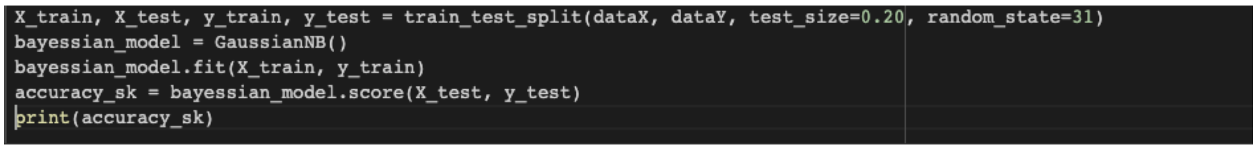
```
model.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=keras.optimizers.Adam(),
    metrics=["accuracy"]
)
```


input_3 | input: | [(None, 12)] | [(None, 12)]
InputLayer | output:
dense_5 | input: | (None, 12) | (None, 512)
Dense | output:
dense_6 | input: | (None, 512) | (None, 64)
Dense | output:
dense_7 | input: | (None, 64) | (None, 16)
Dense | output:
dense_8 | input: | (None, 16) | (None, 5)
Dense | output:

Results

Gaussian Bayesian Classifier

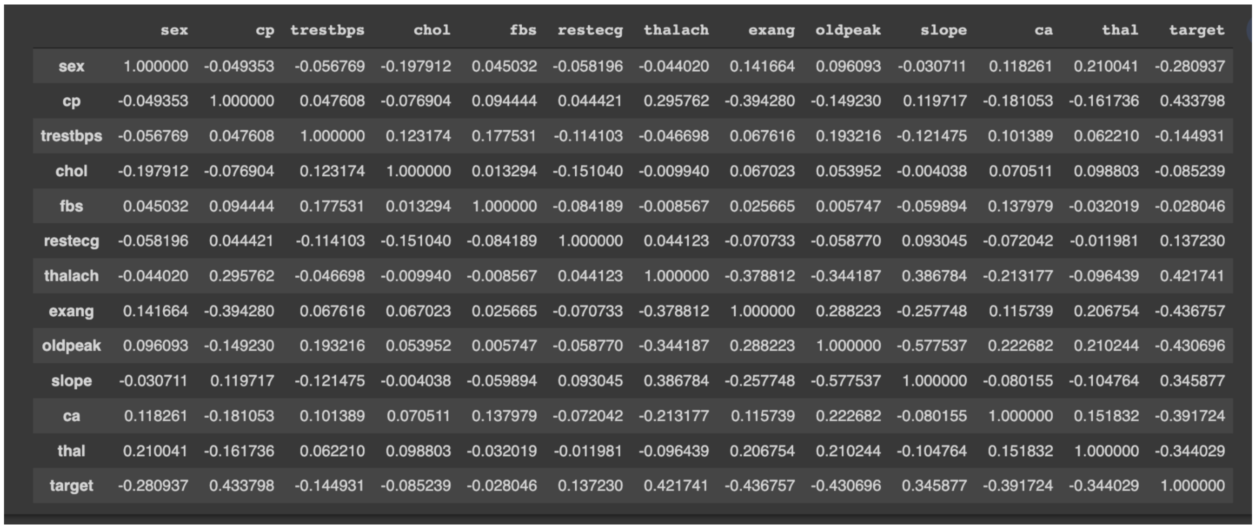
The sklearn implementation of Gaussian Bayesian resulted in an accuracy of 90%.



```
X_train, X_test, y_train, y_test = train_test_split(dataX, dataY, test_size=0.20, random_state=31)
bayessian_model = GaussianNB()
bayessian_model.fit(X_train, y_train)
accuracy_sk = bayessian_model.score(X_test, y_test)
print(accuracy_sk)
```

Sklearn Gaussian Bayessian Implementation

Similarly, our own implementation resulted in an accuracy of 90%, which makes sense because the process of doing Gaussian Bayesian is singular. This confirmed that our Gaussian Bayesian was done correctly because we were using the same training and testing data for both the sklearn version and our own. Lastly, our Gaussian Bayesian performed the best out of all the other methods. I believe this is due to the fact that our data was very close to normal.



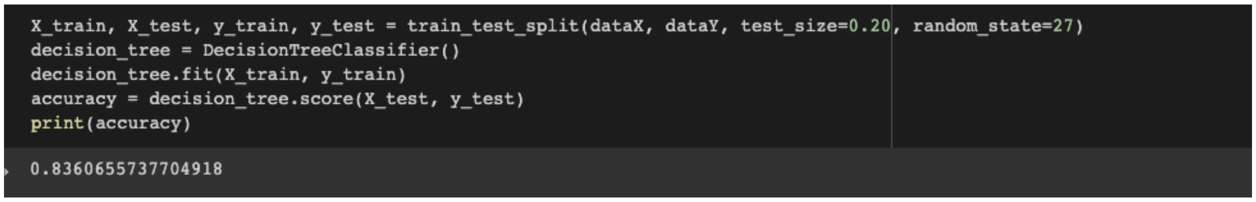
	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
sex	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141664	0.096093	-0.030711	0.118261	0.210041	-0.280937
cp	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394280	-0.149230	0.119717	-0.181053	-0.161736	0.433798
trestbps	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067616	0.193216	-0.121475	0.101389	0.062210	-0.144931
chol	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067023	0.053952	-0.004038	0.070511	0.098803	-0.085239
fbs	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025665	0.005747	-0.059894	0.137979	-0.032019	-0.028046
restecg	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070733	-0.058770	0.093045	-0.072042	-0.011981	0.137230
thalach	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378812	-0.344187	0.386784	-0.213177	-0.096439	0.421741
exang	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000000	0.288223	-0.257748	0.115739	0.206754	-0.436757
oldpeak	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.288223	1.000000	-0.577537	0.222682	0.210244	-0.430696
slope	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.257748	-0.577537	1.000000	-0.080155	-0.104764	0.345877
ca	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.115739	0.222682	-0.080155	1.000000	0.151832	-0.391724
thal	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.206754	0.210244	-0.104764	0.151832	1.000000	-0.344029
target	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.436757	-0.430696	0.345877	-0.391724	-0.344029	1.000000

Correlation of Features

Another reason for the great performance of Gaussian Bayesian could be due to the fact that all the features have very little correlation with each as seen in the figure above. Gaussian Bayesina performs better when the features are independent of each other.

Decision Tree

The sklearn performance of the decision tree was around 83%; the implementation is shown below.



```
X_train, X_test, y_train, y_test = train_test_split(dataX, dataY, test_size=0.20, random_state=27)
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
accuracy = decision_tree.score(X_test, y_test)
print(accuracy)
```

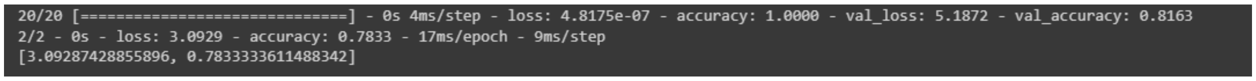
```
0.8360655737704918
```

Sklearn Decision Implementation

Our own decision tree had an accuracy of 72%. I believe we can make changes to our implementation that could improve accuracy. One way could be by adjusting min_split_count; post-presentation testing I noticed that increasing the min_split_count actually improved accuracy.

Neural Network

The tensorflow implementation of the neural network algorithm ultimately achieved around 78% accuracy. We worked around initially with various model hyperparameters, but ultimately chose to work with the model described earlier as it resulted in the highest accuracy and consistency. If we had more time to research the pros and cons behind the various hyperparameters, we could have resulted in a more accurate neural network model.

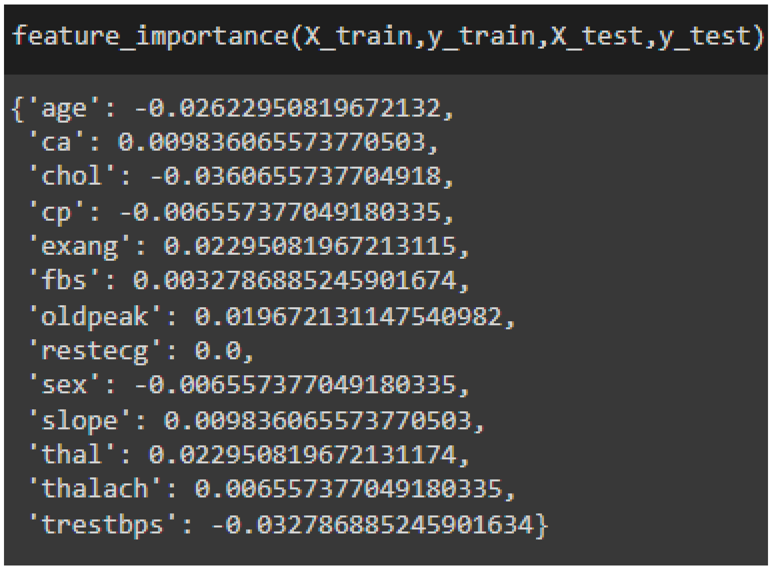


```
20/20 [==============================] - 0s 4ms/step - loss: 4.8175e-07 - accuracy: 1.0000 - val_loss: 5.1872 - val_accuracy: 0.8163
2/2 - 0s - loss: 3.0929 - accuracy: 0.7833 - 17ms/epoch - 9ms/step
[3.09287428855896, 0.7833333611488342]
```

Our own implementation also ended around 78% accuracy. Since this was a very naive implementation of gradient descent, we believe that more research and implementation of more standard and contemporary models would have resulted in a better suited algorithm.

Discussion

Analyzing the results of the various methods, we see Gaussian Bayes as the best suited algorithms to solve the problem at hand. This refuted our original preconceptions that a neural network could always be tuned into the best suited model with the right hyperparameters. Gaussian Bayes is especially useful when working with data where features are quite independent from one another. Looking more in-depth at the correlation values between different features in our dataset, there was no correlation value greater than 0.5. This confirmed that the features, although unexpected, were quite independent from one another. The neural network model also turned out to be much more subpar even compared to the decision tree algorithm. Running feature importance on our models resulted in a high importance value around thal, exang, and oldpeak. If more work could be done on the model, it's possible that refining the scope of the feature set could result in a finer accuracy. Ultimately, in a product standpoint, we confirmed that Gaussian Bayes is the best algorithm for the current data set and problem.



```
feature_importance(X_train,y_train,X_test,y_test)

{'age': -0.02622950819672132,
 'ca': 0.009836065573770503,
 'chol': -0.0360655737704918,
 'cp': -0.006557377049180335,
 'exang': 0.022950819672131115,
 'fbs': 0.0032786885245901674,
 'oldpeak': 0.019672131147540982,
 'restecg': 0.0,
 'sex': -0.006557377049180335,
 'slope': 0.009836065573770503,
 'thal': 0.022950819672131174,
 'thalach': 0.006557377049180335,
 'trestbps': -0.032786885245901634}
```