Fetal Health Prediction

Anurag Dinesh Karmarkar

Table Of Content

1. Introduction

1.1 Report Overview

1.2 Overview of Methodology

2. Predictive Modelling

2.1 Feature Selection

2.2 Model Fitting & Tuning

2.3 Model Comparison

3. Critique & Limitations

4. Summary & Conclusions

4.1 Project Summary

4.2 Summary of Findings

4.3 Conclusions

References

Introduction

Report Overview

Child Mortality is basially death of new born child under age of 5. The rate of child mortality is basically probability of dying between birth and exactly five years of age expressed per 1,000 live births. In this task we will be using fetal health dataset which has 2126 records of health features extracted from carditogram monitoring. The health features used are Heart rate ,fetal movement,uterine contractions,Decelerations and abnormal short-term variability.Thus, we wil be using above mentioned features to predict the fetal health and try to avoid further child and maternal mortality.

Overview of Methodology

The first step of predictive modelling is basically bifurcating the features and target from data frames.

After setting target and features we must scale the features using standard min-max scalar.

The purpose of scaling is to calculate the base distance between data features.

The train-test split mechanism should be used to split training datasets and testing datasets.

Train-Test splitting is must as its used to estimate the performance of machine learning models while predicting the target.

Feature selection is the next process after train and test datasets are separated.

Feature selection is basically selecting informative features which can give good results while prediction.

After feature selection, the main process is to finalize the machine learning model, the initial stage before Modelling is to analyse the problem statement and to decide problem type. Thus, we are dealing with classification problem

Machine Learning models used for Classification: -

1)KNN - K nearest neighbour algorithm is simplest algorithm which use data and classify new data points based on similarity measures (e.g., distance function)

2)Logistic Regression - Logistic regression is a classification algorithm. It is used to predict a binary outcome based on a set of independent variables.

3)Decision Tree - Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks.

4)Random Forest - Random Forest is combination of many decision trees which are made by bootstrapping of features to do prediction.

5)Naive Bayes - Naive bayes classifiers are family of probabilistic classifier's which are based on Bayes theorem.

6)Support Vector Classifier - The SVC is a wrapper around the libsvm library and supports different kernels while LinearSVC is based on liblinear and only supports a linear kernel.

Data Preprocessing

In [1]:
```
#Importing required packages
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from scipy import stats
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

In [2]:
```
#Creating The dataset variable
fetal_p = 'Data_group21.csv'
```

In [3]:
```
#Reading the dataset and displaying the head
fetal = pd.read_csv(fetal_p)
fetal.head(10)
```

Out[3]:

	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	severe_dece
0	120.0	0.000	0.0	0.000	0.000	
1	132.0	0.006	0.0	0.006	0.003	
2	133.0	0.003	0.0	0.008	0.003	
3	134.0	0.003	0.0	0.008	0.003	
4	132.0	0.007	0.0	0.008	0.000	
5	134.0	0.001	0.0	0.010	0.009	
6	134.0	0.001	0.0	0.013	0.008	
7	122.0	0.000	0.0	0.000	0.000	
8	122.0	0.000	0.0	0.002	0.000	
9	122.0	0.000	0.0	0.003	0.000	

10 rows × 22 columns

In [4]:
```
#A small code to look at the type of data stored in each and every column
print(fetal.shape)
for i in range(22):
    print(type(fetal.iloc[0,i]))
```

```
(2126, 22)
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

In [5]:
```
# Because we have all data as numerical hence describing the dataset Transposing for a better view
fetal.describe().T
```

Out[5]:

	count	mean	std	min	
baseline value	2126.0	133.303857	9.840844	106.0	126
accelerations	2126.0	0.003178	0.003866	0.0	0
fetal_movement	2126.0	0.009481	0.046666	0.0	0
uterine_contractions	2126.0	0.004366	0.002946	0.0	0
light_decelerations	2126.0	0.001889	0.002960	0.0	0
severe_decelerations	2126.0	0.000003	0.000057	0.0	0
prolongued_decelerations	2126.0	0.000159	0.000590	0.0	0
abnormal_short_term_variability	2126.0	46.990122	17.192814	12.0	32
mean_value_of_short_term_variability	2126.0	1.332785	0.883241	0.2	0
percentage_of_time_with_abnormal_long_term_variability	2126.0	9.846660	18.396880	0.0	0
mean_value_of_long_term_variability	2126.0	8.187629	5.628247	0.0	4
histogram_width	2126.0	70.445908	38.955693	3.0	37
histogram_min	2126.0	93.579492	29.560212	50.0	67
histogram_max	2126.0	164.025400	17.944183	122.0	152
histogram_number_of_peaks	2126.0	4.068203	2.949386	0.0	2
histogram_number_of_zeroes	2126.0	0.323612	0.706059	0.0	0
histogram_mode	2126.0	137.452023	16.381289	60.0	129
histogram_mean	2126.0	134.610536	15.593596	73.0	125
histogram_median	2126.0	138.090310	14.466589	77.0	129
histogram_variance	2126.0	18.808090	28.977636	0.0	2
histogram_tendency	2126.0	0.320320	0.610829	-1.0	0
fetal_health	2126.0	1.304327	0.614377	1.0	1

In [6]:
```
#checking dataset for the null values
fetal.isnull().sum()
```

Out[6]:
```
baseline value                                            0
accelerations                                             0
fetal_movement                                            0
uterine_contractions                                      0
light_decelerations                                       0
severe_decelerations                                      0
prolongued_decelerations                                  0
abnormal_short_term_variability                           0
mean_value_of_short_term_variability                      0
percentage_of_time_with_abnormal_long_term_variability    0
mean_value_of_long_term_variability                       0
histogram_width                                           0
histogram_min                                             0
histogram_max                                             0
histogram_number_of_peaks                                 0
histogram_number_of_zeroes                                0
histogram_mode                                            0
histogram_mean                                            0
histogram_median                                          0
histogram_variance                                        0
histogram_tendency                                        0
fetal_health                                              0
dtype: int64
```

In [7]:
```
#checking dataset for the na values(because its all numeric)
fetal.isna().sum()
```

Out[7]:
```
baseline value                                            0
accelerations                                             0
fetal_movement                                            0
uterine_contractions                                      0
light_decelerations                                       0
severe_decelerations                                      0
prolongued_decelerations                                  0
abnormal_short_term_variability                           0
mean_value_of_short_term_variability                      0
percentage_of_time_with_abnormal_long_term_variability    0
mean_value_of_long_term_variability                       0
histogram_width                                           0
histogram_min                                             0
histogram_max                                             0
histogram_number_of_peaks                                 0
histogram_number_of_zeroes                                0
histogram_mode                                            0
histogram_mean                                            0
histogram_median                                          0
histogram_variance                                        0
histogram_tendency                                        0
fetal_health                                              0
dtype: int64
```

In [8]:
```
# Getting The type of values in our target variable
fetal['fetal_health'].value_counts()
```

Out[8]:
```
1.0    1655
2.0     295
3.0     176
Name: fetal_health, dtype: int64
```

In [9]:
```
#As we are planing to do classification hence changing the target from numeric to labels
fetal['fetal_health'].replace({1.0 : "Normal" , 2.0: "Suspect" , 3.0 :"Pathalogical"},inplace = True)
fetal['fetal_health'].value_counts()
```

Out[9]:
```
Normal          1655
Suspect          295
Pathalogical     176
Name: fetal_health, dtype: int64
```

Predictive Modelling

Scaling & Feature Selection

Feature selection is also called as variable selection. It is the process of selecting subset of relevant attributes which are in use for model construction.

The purpose of feature selection is to do simplification of models to make them easier to interpret by end users, it shortens the training time of model as unwanted features are separated and enhances generalisation by reducing overfitting.

Here, we have used k-best feature selection method, it selects features according to their highest score. Thus, the feature with higher score will be selected and rest features will be expelled.

In [10]:
```
# assigning values to features as X and target as y
fetal_features=fetal.drop(["fetal_health"],axis=1)
fetal_target=fetal["fetal_health"]

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

#Set up a Min Max scaler for the features
col_names = list(fetal_features.columns)
mm_scaler = preprocessing.MinMaxScaler()
fetal_features_scaled= mm_scaler.fit_transform(fetal_features)
fetal_features_scaled = pd.DataFrame(fetal_features_scaled, columns=col_names)
fetal_features_scaled.describe().T
```

Out[10]:

	count	mean	std	min	25%
baseline value	2126.0	0.505627	0.182238	0.0	0.37037
accelerations	2126.0	0.167277	0.203452	0.0	0.00000
fetal_movement	2126.0	0.019710	0.097018	0.0	0.00000
uterine_contractions	2126.0	0.291094	0.196405	0.0	0.13333
light_decelerations	2126.0	0.125964	0.197347	0.0	0.00000
severe_decelerations	2126.0	0.003293	0.057300	0.0	0.00000
prolongued_decelerations	2126.0	0.031703	0.117990	0.0	0.00000
abnormal_short_term_variability	2126.0	0.466535	0.229238	0.0	0.26666
mean_value_of_short_term_variability	2126.0	0.166586	0.129888	0.0	0.07352
percentage_of_time_with_abnormal_long_term_variability	2126.0	0.108205	0.202164	0.0	0.00000
mean_value_of_long_term_variability	2126.0	0.161492	0.111011	0.0	0.09073
histogram_width	2126.0	0.381050	0.220089	0.0	0.19209
histogram_min	2126.0	0.399812	0.271195	0.0	0.15596
histogram_max	2126.0	0.362288	0.154691	0.0	0.25862
histogram_number_of_peaks	2126.0	0.226011	0.163855	0.0	0.11111
histogram_number_of_zeroes	2126.0	0.032361	0.070606	0.0	0.00000
histogram_mode	2126.0	0.609858	0.128987	0.0	0.54330
histogram_mean	2126.0	0.565234	0.143061	0.0	0.47706
histogram_median	2126.0	0.560462	0.132721	0.0	0.47706
histogram_variance	2126.0	0.069919	0.107724	0.0	0.00743
histogram_tendency	2126.0	0.660160	0.305414	0.0	0.50000

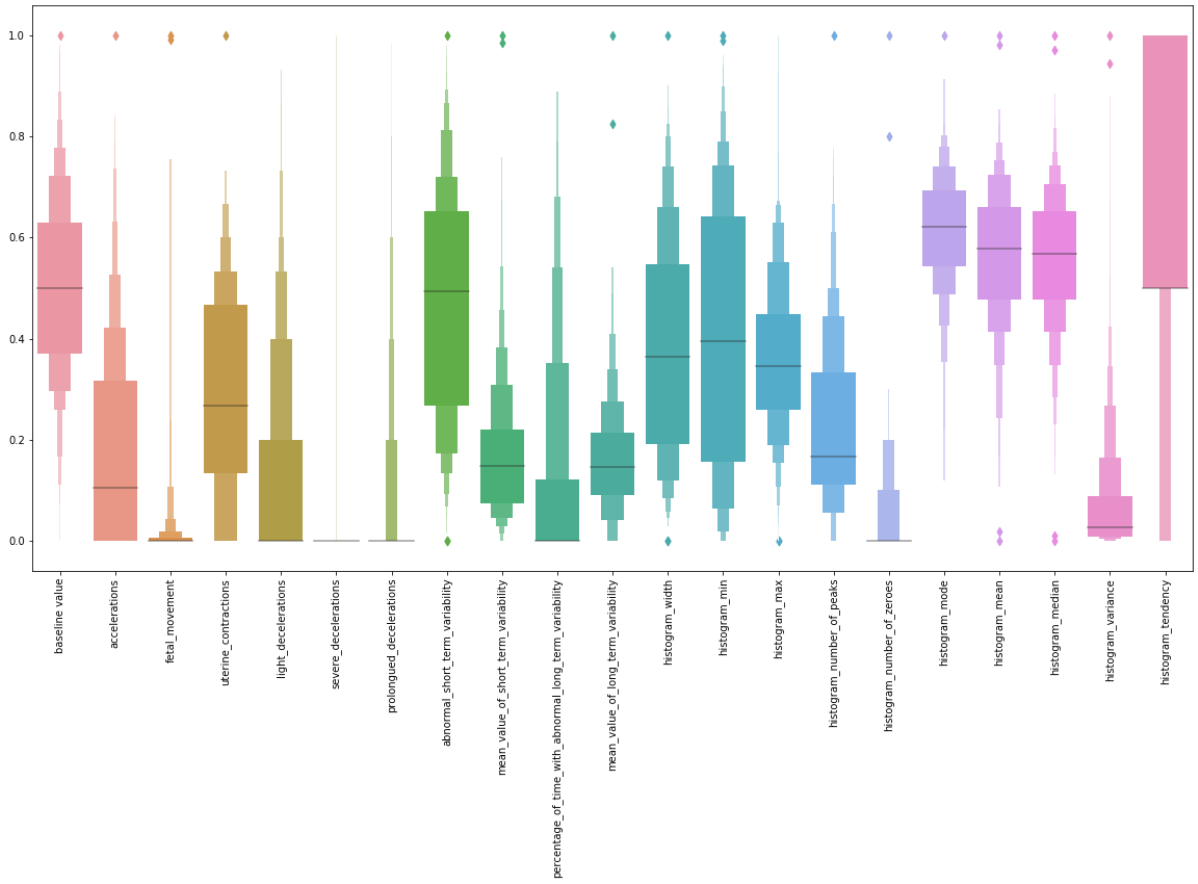
In [11]:
```
fetal_features_scaled.head(10)
```

Out[11]:

	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	severe_dece
0	0.259259	0.000000	0.0	0.000000	0.000000	
1	0.481481	0.315789	0.0	0.400000	0.200000	
2	0.500000	0.157895	0.0	0.533333	0.200000	
3	0.518519	0.157895	0.0	0.533333	0.200000	
4	0.481481	0.368421	0.0	0.533333	0.000000	
5	0.518519	0.052632	0.0	0.666667	0.600000	
6	0.518519	0.052632	0.0	0.866667	0.533333	
7	0.296296	0.000000	0.0	0.000000	0.000000	
8	0.296296	0.000000	0.0	0.133333	0.000000	
9	0.296296	0.000000	0.0	0.200000	0.000000	

10 rows × 21 columns

In [12]:
```
#looking at the scaled features
# We can see all our features are in range for 0-1
plt.figure(figsize=(20,10))
sns.boxenplot(data = fetal_features_scaled)
plt.xticks(rotation=90)
plt.show()
```



In [13]:
```
#Determining the scores for each of the three features for prediction
from sklearn.feature_selection import SelectKBest
from sklearn import feature_selection as fs



#Input columns
X = fetal_features_scaled
#Target column i.e Activity label
y = fetal_target



#Applying SelectKBest class to extract feature scores
bestfeatures = SelectKBest(fs.f_classif)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']
```

In [14]:
```
featureScores.sort_values('Score', ascending=False)
```

Out[14]:

	Specs	Score
6	prolongued_decelerations	505.853206
9	percentage_of_time_with_abnormal_long_term_var...	345.156385
7	abnormal_short_term_variability	343.820419
17	histogram_mean	297.625497
16	histogram_mode	275.117696
18	histogram_median	248.772237
1	accelerations	196.027523
19	histogram_variance	150.796849
0	baseline value	140.621076
8	mean_value_of_short_term_variability	119.882006
3	uterine_contractions	93.715743
12	histogram_min	87.340503
10	mean_value_of_long_term_variability	70.174093
4	light_decelerations	66.864754
11	histogram_width	55.088241
20	histogram_tendency	44.542294
5	severe_decelerations	28.448156
14	histogram_number_of_peaks	12.104834
2	fetal_movement	11.679797
13	histogram_max	2.464923
15	histogram_number_of_zeroes	2.196373

In [15]:
```
fetal_features_scaled_input =fetal_features_scaled[['prolongued_decelerations'
,'percentage_of_time_with_abnormal_long_term_variability',
                                                    'abnormal_short_term_varia
bility','histogram_mean','histogram_mode',
                                                    'histogram_median','accele
rations','histogram_variance','baseline value',
                                                    'mean_value_of_short_term_
variability']]
```

In [16]:
```
fetal_features_scaled_input.head(10)
```

Out[16]:

	prolongued_decelerations	percentage_of_time_with_abnormal_long_term_variability	abnormal_sh
0	0.0	0.472527	
1	0.0	0.000000	
2	0.0	0.000000	
3	0.0	0.000000	
4	0.0	0.000000	
5	0.4	0.000000	
6	0.6	0.000000	
7	0.0	0.065934	
8	0.0	0.054945	
9	0.0	0.065934	

Model Fitting & Tuning

We have used the classifiers such as k Nearest Neighbours, Decision Tree, Random Forest, Stochastic Gradient Boosting and Naive Bayes. We will select the best model after tuning the hyperparameters of all the selected models. We will use the GridSearchCV() function to select the best values for the hyper parameters.

KNN Fitting and Hyperparameter Tuning

The hyper parameters tuned for KNN classifier are n_neighbors and p (metric).

In [17]:
```
cv_method = RepeatedStratifiedKFold(n_splits=5,
                                    n_repeats=3,
                                    random_state=999)
params_KNN = {'n_neighbors': [1,2, 3,4, 5,6, 7],
              'p': [1, 2, 5]}
gs_KNN = GridSearchCV(estimator=KNeighborsClassifier(),
                      param_grid=params_KNN,
                      cv=cv_method,
                      verbose=1,  # verbose: the higher, the more messages
                      scoring='accuracy',
                      return_train_score=True)
gs_KNN.fit(fetal_features_scaled_input, fetal_target);
```

```
Fitting 15 folds for each of 21 candidates, totalling 315 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
[Parallel(n_jobs=1)]: Done 315 out of 315 | elapsed:   48.0s finished
```

In [18]:
```
gs_KNN.best_params_
```

Out[18]:
```
{'n_neighbors': 1, 'p': 1}
```

In [19]:
```
gs_KNN.best_score_
```

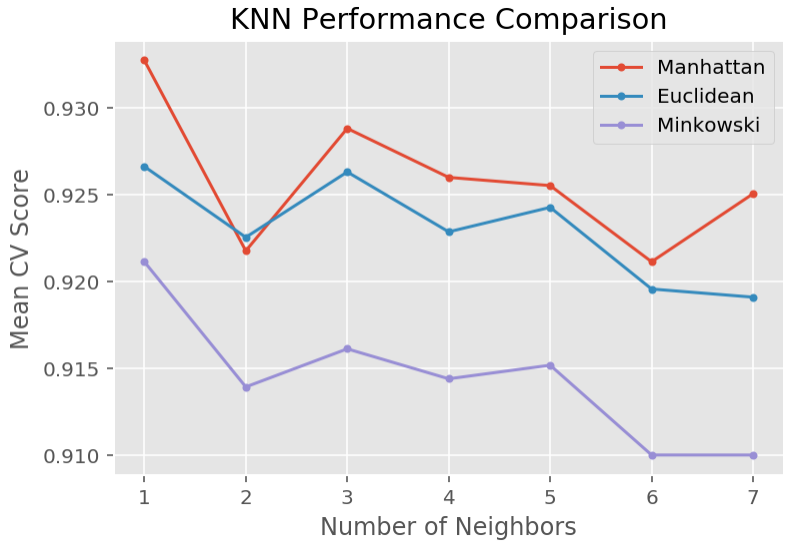
Out[19]:
```
0.9327382859246984
```

In [20]:
```
results_KNN = pd.DataFrame(gs_KNN.cv_results_['params'])
results_KNN['test_score'] = gs_KNN.cv_results_['mean_test_score']
results_KNN['metric'] = results_KNN['p'].replace([1,2,5], ["Manhattan", "Eucli
dean", "Minkowski"])
```

In [21]:
```
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
plt.style.use("ggplot")

for i in ["Manhattan", "Euclidean", "Minkowski"]:
    temp = results_KNN[results_KNN['metric'] == i]
    plt.plot(temp['n_neighbors'], temp['test_score'], marker = '.', label = i)

plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel("Mean CV Score")
plt.title("KNN Performance Comparison")
plt.show()
```



We have found that the best parameters for KNN classifier are n_neighbors = 1 and p = 1 but n_neighbors=1 is not a good for modelling as it may lead to Overfitting hence we have choosen the next best parameters viz. n_neighbors = 3 and p = 1

Decision Tree Fitting & HyperParameter Tunning

The hyper parameters tuned for Decision Tree classifier are criterion, max_depth and min_samples_split.

In [22]:
```
df_classifier = DecisionTreeClassifier(random_state=999)

params_DT = {'criterion': ['gini', 'entropy'],
             'max_depth': [1, 2, 3, 4, 5, 6, 7, 8],
             'min_samples_split': [3,4,5]}

gs_DT = GridSearchCV(estimator=df_classifier,
                     param_grid=params_DT,
                     cv=cv_method,
                     verbose=1,
                     scoring='accuracy')

gs_DT.fit(fetal_features_scaled_input, fetal_target);
```

```
Fitting 15 folds for each of 48 candidates, totalling 720 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
[Parallel(n_jobs=1)]: Done 720 out of 720 | elapsed:    8.1s finished
```

In [23]:
```
gs_DT.best_params_
```

Out[23]:
```
{'criterion': 'entropy', 'max_depth': 7, 'min_samples_split': 4}
```

In [24]:
```
gs_DT.best_score_
```

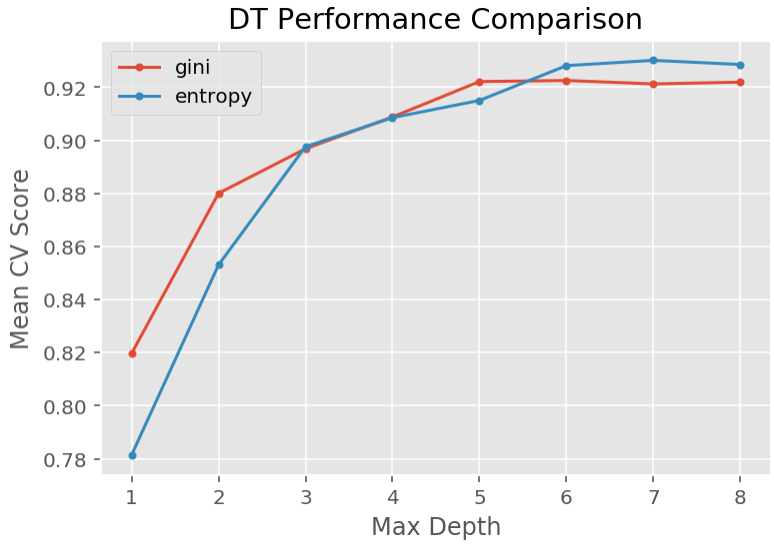
Out[24]:
```
0.9302292184479425
```

In [25]:
```
results_DT = pd.DataFrame(gs_DT.cv_results_['params'])
results_DT['test_score'] = gs_DT.cv_results_['mean_test_score']
results_DT.columns
```

Out[25]:
```
Index(['criterion', 'max_depth', 'min_samples_split', 'test_score'], dtype='o
bject')
```

In [26]:
```
for i in ['gini', 'entropy']:
    temp = results_DT[results_DT['criterion'] == i]
    temp_average = temp.groupby('max_depth').agg({'test_score': 'mean'})
    plt.plot(temp_average, marker = '.', label = i)


plt.legend()
plt.xlabel('Max Depth')
plt.ylabel("Mean CV Score")
plt.title("DT Performance Comparison")
plt.show()
```



We have found that the best parameters for Decision Tree classifier are criterion = entropy, max_depth = 7 and min_samples_split = 4

Naive Bayes Model Fitting & HyperParameter tuning

The hyper parameter tuned for KNN classifier is var_smoothing.

In [27]:
```
np.random.seed(999)

nb_classifier = GaussianNB()

params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}

gs_NB = GridSearchCV(estimator=nb_classifier,
                     param_grid=params_NB,
                     cv=cv_method,
                     verbose=1,
                     scoring='accuracy')

gs_NB.fit(fetal_features_scaled_input, fetal_target);
```

```
Fitting 15 folds for each of 100 candidates, totalling 1500 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
[Parallel(n_jobs=1)]: Done 1500 out of 1500 | elapsed:    9.7s finished
```

In [28]:
```
gs_NB.best_params_
```

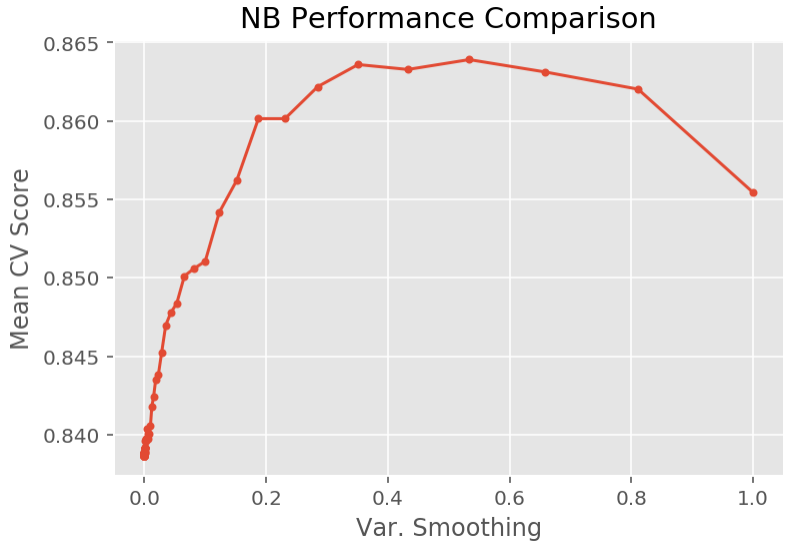
Out[28]:
```
{'var_smoothing': 0.533669923120631}
```

In [29]:
```
gs_NB.best_score_
```

Out[29]:
```
0.8639079443984167
```

In [30]:
```
results_NB = pd.DataFrame(gs_NB.cv_results_['params'])
results_NB['test_score'] = gs_NB.cv_results_['mean_test_score']
```

In [31]:
```
plt.plot(results_NB['var_smoothing'], results_NB['test_score'], marker = '.')
plt.xlabel('Var. Smoothing')
plt.ylabel("Mean CV Score")
plt.title("NB Performance Comparison")
plt.show()
```



The best value for var_smoothing is found to be 0.533669923120631

Random Forest Model Fitting & HyperParamter tuning

The hyper parameters tuned for Random Forest classifier are n_estimators and max_features.The n_estimators are set to logarithmic values and max_features refers to the function used for calculation.

In [32]:
```
# define models and parameters
model = RandomForestClassifier()
n_estimators = [10, 100, 1000]
max_features = ['sqrt', 'log2']
# define grid search
grid = dict(n_estimators=n_estimators,max_features=max_features)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,
scoring='accuracy',error_score=0)
grid_result = grid_search.fit(fetal_features_scaled_input, fetal_target)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_
))
```

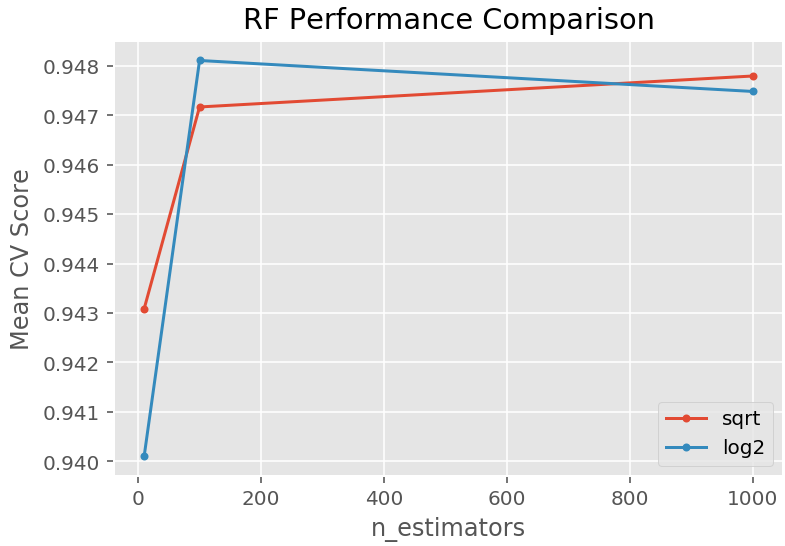
```
Best: 0.948110 using {'max_features': 'log2', 'n_estimators': 100}
```

In [33]:
```
results_RF = pd.DataFrame(grid_result.cv_results_['params'])
results_RF['test_score'] = grid_result.cv_results_['mean_test_score']
results_RF.columns
```

Out[33]:
```
Index(['max_features', 'n_estimators', 'test_score'], dtype='object')
```

In [34]:
```
for i in ['sqrt', 'log2']:
    temp = results_RF[results_RF['max_features'] == i]
    temp_average = temp.groupby('n_estimators').agg({'test_score': 'mean'})
    plt.plot(temp_average, marker = '.', label = i)


plt.legend()
plt.xlabel('n_estimators')
plt.ylabel("Mean CV Score")
plt.title("RF Performance Comparison")
plt.show()
```



We have found that the best parameters for Random Forest classifier are max_features = log2 and n_estimators = 1000

Support Vector Classifier modelling & Parameter Tuning

The hyper parameters tuned for SVC classifier are kernel, C and gamma.

In [35]:
```
# define model and parameters
model = SVC()
kernel = ['poly', 'rbf', 'sigmoid']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']
# define grid search
grid = dict(kernel=kernel,C=C,gamma=gamma)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,
scoring='accuracy',error_score=0)
grid_result = grid_search.fit(fetal_features_scaled_input, fetal_target)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_
))
```

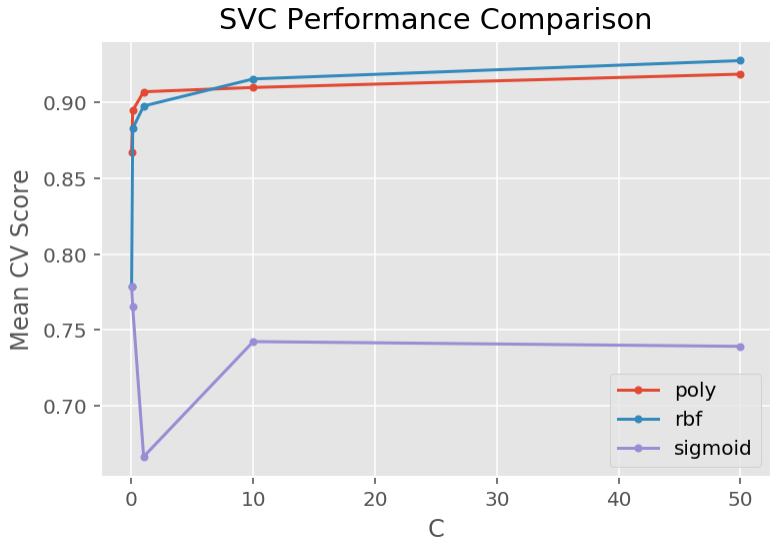
```
Best: 0.927562 using {'C': 50, 'gamma': 'scale', 'kernel': 'rbf'}
```

In [36]:
```
results_SVC = pd.DataFrame(grid_result.cv_results_['params'])
results_SVC['test_score'] = grid_result.cv_results_['mean_test_score']
results_SVC.columns
```

Out[36]:
```
Index(['C', 'gamma', 'kernel', 'test_score'], dtype='object')
```

In [37]:
```
for i in ['poly', 'rbf', 'sigmoid']:
    temp = results_SVC[results_SVC['kernel'] == i]
    temp_average = temp.groupby('C').agg({'test_score': 'mean'})
    plt.plot(temp_average, marker = '.', label = i)


plt.legend()
plt.xlabel('C')
plt.ylabel("Mean CV Score")
plt.title("SVC Performance Comparison")
plt.show()
```



The hyper parameters tuned for SVC classifier are C = 50, gamma = scale and kernel = rbf.

Model Comparision

We have used the models with best tuned parameters and created pipelines for the best model each classifier. We have split the data into training and test data and evaluated the classifier based on the training and test data. The models' are compaired on the basis of their Cross validation scores. We have also printed the Confusion Matrix and Classification report for each Algorithm.

In [38]:
```
# spliting test and training sets
# looking at the shapes of the splitted data sets
X_train, X_test, y_train,y_test = train_test_split(fetal_features_scaled_input
,fetal_target,test_size=0.3,random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[38]:
```
((1488, 10), (638, 10), (1488,), (638,))
```

In [39]:
```
#A quick model selection process
#pipelines of models( it is short was to fit and pred)
pipeline_lr=Pipeline([('lr_classifier',LogisticRegression(random_state=42))])

pipeline_knn = Pipeline([('knn_classifier',KNeighborsClassifier(3))])

pipeline_dt=Pipeline([ ('dt_classifier',DecisionTreeClassifier(criterion = 'en
tropy' , max_depth = 7 , min_samples_split = 4 ,random_state=42))])

pipeline_rf=Pipeline([('rf_classifier',RandomForestClassifier( max_features =
'sqrt' , n_estimators = 100))])

pipeline_nb = Pipeline([('nb_classifier',GaussianNB(var_smoothing=0.5336699
))])

pipeline_svc = Pipeline([('svc_classifier',SVC(C= 50, gamma = 'scale', kernel
= 'rbf'))])

# List of all the pipelines
pipelines = [pipeline_lr,pipeline_knn, pipeline_dt, pipeline_rf,pipeline_nb,pi
peline_svc]

# Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {0: 'Logistic Regression',1:'K Nearest Neighbors' ,2: 'Decision Tr
ee', 3: 'RandomForest', 4: "Naive bayes" , 5:"Support Vector Classifier "}

# Generating the Classification Report & Confusion Matrix for each Algorithm
for i,pipe in enumerate(pipelines):
    fit = pipe.fit(X_train,y_train)
    y_pre = fit.predict(X_test)
    print("Confusion matrix for :" + pipe_dict[i] +"\n")

    print(confusion_matrix(y_test, y_pre))

    print("\nClassification Report for :" + pipe_dict[i] +"\n")

    print(classification_report(y_test,y_pre))


#cross validation on accuracy
print("\nCross Validation Results for our Algorithms")
cv_results_accuracy = []
for i, model in enumerate(pipelines):
    cv_score = cross_val_score(model, X_train,y_train, cv=10 )
    cv_results_accuracy.append(cv_score)
    print("%s: %f " % (pipe_dict[i], cv_score.mean()))
```

```
Confusion matrix for :Logistic Regression

[[479   2  15]
 [  6  30   5]
 [ 37   5  59]]

Classification Report for :Logistic Regression

              precision    recall  f1-score   support

      Normal       0.92      0.97      0.94       496
Pathalogical       0.81      0.73      0.77        41
     Suspect       0.75      0.58      0.66       101

    accuracy                           0.89       638
   macro avg       0.83      0.76      0.79       638
weighted avg       0.88      0.89      0.88       638

Confusion matrix for :K Nearest Neighbors

[[478   1  17]
 [  3  37   1]
 [ 21   5  75]]

Classification Report for :K Nearest Neighbors

              precision    recall  f1-score   support

      Normal       0.95      0.96      0.96       496
Pathalogical       0.86      0.90      0.88        41
     Suspect       0.81      0.74      0.77       101

    accuracy                           0.92       638
   macro avg       0.87      0.87      0.87       638
weighted avg       0.92      0.92      0.92       638

Confusion matrix for :Decision Tree

[[482   6   8]
 [  3  38   0]
 [ 28   1  72]]

Classification Report for :Decision Tree

              precision    recall  f1-score   support

      Normal       0.94      0.97      0.96       496
Pathalogical       0.84      0.93      0.88        41
     Suspect       0.90      0.71      0.80       101

    accuracy                           0.93       638
   macro avg       0.89      0.87      0.88       638
weighted avg       0.93      0.93      0.93       638

Confusion matrix for :RandomForest

[[481   4  11]
```

```
 [  2  39   0]
 [ 20   1  80]]
```

Classification Report for :RandomForest

	precision	recall	f1-score	support
Normal	0.96	0.97	0.96	496
Pathalogical	0.89	0.95	0.92	41
Suspect	0.88	0.79	0.83	101
accuracy			0.94	638
macro avg	0.91	0.90	0.90	638
weighted avg	0.94	0.94	0.94	638

Confusion matrix for :Naive bayes

```
[[455   0  41]
 [  6  21  14]
 [ 23   0  78]]
```

Classification Report for :Naive bayes

	precision	recall	f1-score	support
Normal	0.94	0.92	0.93	496
Pathalogical	1.00	0.51	0.68	41
Suspect	0.59	0.77	0.67	101
accuracy			0.87	638
macro avg	0.84	0.73	0.76	638
weighted avg	0.89	0.87	0.87	638

Confusion matrix for :Support Vector Classifier

```
[[464   2  30]
 [  2  38   1]
 [ 14   3  84]]
```

Classification Report for :Support Vector Classifier

	precision	recall	f1-score	support
Normal	0.97	0.94	0.95	496
Pathalogical	0.88	0.93	0.90	41
Suspect	0.73	0.83	0.78	101
accuracy			0.92	638
macro avg	0.86	0.90	0.88	638
weighted avg	0.92	0.92	0.92	638

Cross Validation Results for our Algorithms
Logistic Regression: 0.875680
K Nearest Neighbors: 0.926769
Decision Tree: 0.907968
RandomForest: 0.934165

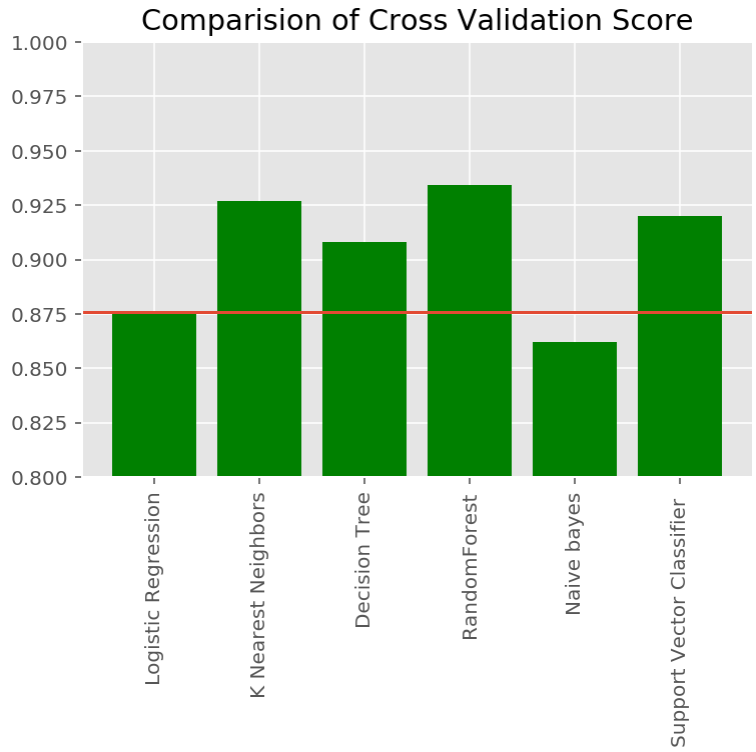
```
Naive bayes: 0.862257
Support Vector Classifier : 0.920053
```

In [40]:
```
#saving the mean cross validation results for each algorithm for easy of inter
pretation and visulaization
cv_results_df = {'Algorithm' : ['Logistic Regression','K Nearest Neighbors' ,
'Decision Tree','RandomForest',
                                "Naive bayes" ,"Support Vector Classifier "],
                     'Score' : [cv_results_accuracy[0].mean(),cv_results_accurac
y[1].mean(),cv_results_accuracy[2].mean(),
                                   cv_results_accuracy[3].mean(),cv_results_accurac
y[4].mean(),cv_results_accuracy[5].mean()]}
cv_results_df = pd.DataFrame(cv_results_df)
cv_results_df
```

Out[40]:

	Algorithm	Score
0	Logistic Regression	0.875680
1	K Nearest Neighbors	0.926769
2	Decision Tree	0.907968
3	RandomForest	0.934165
4	Naive bayes	0.862257
5	Support Vector Classifier	0.920053

In [41]:
```
# A bar graph showing the Cross Validation score of each algorithm
# Logistic regression being the bench mark model
plt.bar(cv_results_df['Algorithm'] , cv_results_df['Score'], color = 'green')
plt.xticks(rotation=90)
plt.title('Comparision of Cross Validation Score')
plt.axhline(y=0.875680)
plt.ylim([0.8,1])
plt.show()
```



We have also calculated the score of Logistic Regression algorithm with the other classifiers to set it as a baseline. The graph shows the comparison of the scores of the classifiers. It can be observed that the Random Forest scores the best out of all the classifiers. Naive Bayes is the lowest among the classifiers as it's score is lower than the baseline score of Logistic Regression.

Paired T-Tests Code Section

Code for kNN

In [42]:
```
# creating code for conducting paired t-tests with all other algorithms except
Logistic Regression
knn_ttest = []
for i in range(1,6):
    if( i != 1 ):
        knn_ttest.append(stats.ttest_rel(cv_results_accuracy[1], cv_results_ac
curacy[i]).pvalue)
knn_ttest
```

Out[42]:
```
[0.007354939442257745,
 0.22006899088808166,
 3.559659968117146e-08,
 0.3108666092252043]
```

In [43]:
```
# saving the results in a pandas dataframe and displaying them for better read
ablity
knn_ttest_df = {'Algorithm' : ['Decision Tree','RandomForest',"Naive bayes" ,
"Support Vector Classifier "],
                    'P-value' : [knn_ttest[0],knn_ttest[1],knn_ttest[2],knn_tte
st[3]],
                'is_Significant' : ['TRUE','FALSE','TRUE','FALSE']}
knn_ttest_df = pd.DataFrame(knn_ttest_df)
knn_ttest_df
```

Out[43]:

	Algorithm	P-value	is_Significant
0	Decision Tree	7.354939e-03	TRUE
1	RandomForest	2.200690e-01	FALSE
2	Naive bayes	3.559660e-08	TRUE
3	Support Vector Classifier	3.108666e-01	FALSE

We observe from the table that the difference between KNN and the following classifiers are statistically significant:

1. Decision Tree

2. Naïve Bayes

Code for Decision Tree

In [44]:
```
# creating code for conducting paired t-tests with all other algorithms except
Logistic Regression
dt_ttest = []
for i in range(1,6):
    if( i != 2 ):
        dt_ttest.append(stats.ttest_rel(cv_results_accuracy[2], cv_results_acc
uracy[i]).pvalue)
dt_ttest
```

Out[44]:
```
[0.007354939442257745,
 0.0022272650054693728,
 3.256730682555172e-05,
 0.2199674291099932]
```

In [45]:
```
# saving the results in a pandas dataframe and displaying them for better read
ablity
dt_ttest_df = {'Algorithm' : ['K Nearest Neighbors','RandomForest',"Naive baye
s" ,"Support Vector Classifier "],
                   'p-value' : [dt_ttest[0],dt_ttest[1],dt_ttest[2],dt_ttest[3
]],
               'is_Significant' : ['TRUE','TRUE','TRUE','FALSE']}
dt_ttest_df = pd.DataFrame(dt_ttest_df)
dt_ttest_df
```

Out[45]:

	Algorithm	p-value	is_Significant
0	K Nearest Neighbors	0.007355	TRUE
1	RandomForest	0.002227	TRUE
2	Naive bayes	0.000033	TRUE
3	Support Vector Classifier	0.219967	FALSE

We observe from the table that the difference between Decision Tree and the following classifiers are statistically significant:

1. KNN

2. Random Forest

3. Naïve Bayes

Code for Random Forest

In [46]:
```
# creating code for conducting paired t-tests with all other algorithms except
Logistic Regression
rf_ttest = []
for i in range(1,6):
    if( i != 3 ):
        rf_ttest.append(stats.ttest_rel(cv_results_accuracy[3], cv_results_acc
uracy[i]).pvalue)
rf_ttest
```

Out[46]:
```
[0.22006899088808166,
 0.0022272650054693728,
 9.958846223974524e-07,
 0.0982586570808451]
```

In [47]:
```
# saving the results in a pandas dataframe and displaying them for better read
ablity
rf_ttest_df = {'Algorithm' : ['K Nearest Neighbors','Decision Tree',"Naive bay
es" ,"Support Vector Classifier "],
                   'p-value' : [rf_ttest[0],rf_ttest[1],rf_ttest[2],rf_ttest[3
]],
               'is_Significant' : ['FALSE','TRUE','TRUE','TRUE']}
rf_ttest_df = pd.DataFrame(rf_ttest_df)
rf_ttest_df
```

Out[47]:

	Algorithm	p-value	is_Significant
0	K Nearest Neighbors	2.200690e-01	FALSE
1	Decision Tree	2.227265e-03	TRUE
2	Naive bayes	9.958846e-07	TRUE
3	Support Vector Classifier	9.825866e-02	TRUE

We observe from the table that the difference between Random Forest and the following classifiers are statistically significant:

1. Decision Tree

2. Naïve Bayes

3. Support Vector Classifier

Code for Naive bayes

In [48]:
```
# creating code for conducting paired t-tests with all other algorithms except
Logistic Regression
nb_ttest =[]
for i in range(1,6):
    if( i != 4 ):
        nb_ttest.append(stats.ttest_rel(cv_results_accuracy[4], cv_results_acc
uracy[i]).pvalue)
nb_ttest
```

Out[48]:
```
[3.559659968117146e-08,
 3.256730682555172e-05,
 9.958846223974524e-07,
 3.72703461658418e-05]
```

In [49]:
```
# saving the results in a pandas dataframe and displaying them for better read
ablity
nb_ttest_df = {'Algorithm' : ['K Nearest Neighbors','Decision Tree',"RandomFor
est" ,"Support Vector Classifier "],
                'p-value' : [nb_ttest[0],nb_ttest[1],nb_ttest[2],nb_ttest[3]],
              'is_Significant' : ['TRUE','TRUE','TRUE','TRUE']}
nb_ttest_df = pd.DataFrame(nb_ttest_df)
nb_ttest_df
```

Out[49]:

	Algorithm	p-value	is_Significant
0	K Nearest Neighbors	3.559660e-08	TRUE
1	Decision Tree	3.256731e-05	TRUE
2	RandomForest	9.958846e-07	TRUE
3	Support Vector Classifier	3.727035e-05	TRUE

We observe from the table that the difference between Naïve Bayes and the following classifiers are statistically significant:

1. KNN

2. Random Forest

3. Decision Tree

4. Support Vector Classifier

Code for Support Vector Classifier

In [50]:
```
# creating code for conducting paired t-tests with all other algorithms except
Logistic Regression
svc_ttest = []
for i in range(1,6):
    if( i != 5 ):
        svc_ttest.append(stats.ttest_rel(cv_results_accuracy[5], cv_results_ac
curacy[i]).pvalue)
svc_ttest
```

Out[50]:
```
[0.3108666092252043,
 0.2199674291099932,
 0.0982586570808451,
 3.72703461658418e-05]
```

In [51]:
```
# saving the results in a pandas dataframe and displaying them for better read
ablity
svc_ttest_df = {'Algorithm' : ['K Nearest Neighbors','Decision Tree',"RandomFo
rest" ,"Naive Bayes"],
                    'p-value' : [svc_ttest[0],svc_ttest[1],svc_ttest[2],svc_tte
st[3]],
                'is_Significant' : ['FALSE','FALSE','TRUE','TRUE']}
svc_ttest_df = pd.DataFrame(svc_ttest_df)
svc_ttest
```

Out[51]:
```
[0.3108666092252043,
 0.2199674291099932,
 0.0982586570808451,
 3.72703461658418e-05]
```

We observe from the table that the difference between Support Vector Classifier and the following classifiers are statistically significant:

1. Random Forest

2. Naïve Bayes

Critique & Limitations

Strengths about our approach for Classifcation model

**** Data Preperation and Data Cleaning is done***

**** All our variables are Numerical hence all are scaled before modelling***

**** Rigorous feature selection process as well as taking the top 9 features to get a better results***

**** A thorough analysis on model by using are KNN, Decison Tree, Random Forest, Naive Bayes & SVC***

**** Parameter Tuning done for each and every model so that we can get the best results.***

**** Parameter Tuning results shown in form for Graphs so its easy to understand***

**** Added a baseline case by doing a logistic regression so that its easy to compare***

**** Visible graphs for comparison between the models.***

**** Confusion Matrix , Classification Report and Cross Validation done for model comparison***

Limitations & Things to take note of

**** Classification Accuracy depends on the quality of the data if any of the step is missed it may lead to incorrect results.***

**** Classification Algorithms are sensitive to scaling hence its one of the important step***

**** In case of KNN the best Parameter which was suggested was k = 1 and p = 1 but using those parameters may lead to overfitting***

**** Our best prediction comes from Random Forest Algorithm in this case its good but if dataset is too big then the processing time for Random forest can be too much which makes your real time problem solving slow***

**** As we are working on a medical data we have to be very sure about the results before using them and hence it requires more analysis***

Summary & Conclusions

Project Summary

Through this project we wanted to create a sophisticated Machine learning model which can be used to help the Doctors in identifying the health of foetus. In phase one we explored the data set and tried to understand its features with different Visual representations. For this first we cleaned our dataset and then we made sure to see if we have any incorrect or empty values. After that we created the heatmaps, scatter matrix and further other graphs to get a better understanding of our features. In the phase 2 we scale our features then ran a feature selection and found our top 9 parameters. After this we did parameter tuning and found the best hyper parameters for our algorithms. Then we created all the classification models and ran confusion matrix, classification Report and cross validation on all of them. At last, we ran paired T-Tests on our cross validation results to check the significance of our results.

Summary of Findings

As we are working on medical data anything that doesn't account for false negatives is a crime hence Recall is a better measure than precision.

To summarize our findings let's take a look at the Classification reports for all our models.

Algorithm	Normal	Pathological	Suspect
Logistic Regression	97	73	58
KNN	96	90	74
Decision Tree	97	93	71
Random Forest	98	95	80
Naïve Bayes	92	51	77
SVC	94	93	83

We can see that Logistic Regression (Baseline) gives us recall for 97,73 &58 across the target variables. Any model giving us better results than this will be better for our problem.

Taking a look at the above table we can see that with a recall rate of 92,51 & 77 Naïve Bayes is worst algorithm to apply on this dataset. Good news is all the 4 remaining algorithms did better than our baseline and Random Forest in particular was better than rest of them with a recall score of 98, 95 & 80.

Conclusions

World Health Organization (WHO) defines maternal deaths as "the death of a woman while pregnant or within 42 days of termination of pregnancy, irrespective of the duration and site of the pregnancy, from any cause related to or aggravated by the pregnancy or its management but not from accidental or incidental causes." These statistics are better in the developed countries but when it comes to countries which are developing maternal mortality rate is as high as 1200. A lot of these deaths are because of lack of medical facilities. We are suggesting a Machine Learning Classification model which can identify if the foetus is healthy or needs special care. By successfully identifying any complications this early in pregnancy we can save lives of both Mother and Children.

We are actually proud of results which we have got in our first attempt all of our algorithms are gave us a prediction score of almost 90%. Random forest with a score of 0.9341 is our best prediction. When we take a look at our baseline Model of Logistic Regression, we see a 87% correct prediction which confirms that our methodology of feature selection and Scaling of data is correct. With the table given below we can actually infer that Except the Gaussian Naïve Bayes Classifier all the other classifiers are working better than our baseline.

Alorithm	Score	State
Logistic Regression	0.875680	Baseline
K Nearest Neighbors	0.926769	Better
Decision Tree	0.907968	Better
RandomForest	0.934169	Best
Naive bayes	0.862257	Worst
Support Vector Classifier	0.920053	Better

References

1. Kaggle.com. 2020. Fetal Health Classification. [online] Available at: https://www.kaggle.com/andrewmvd/fetal-health-classification (https://www.kaggle.com/andrewmvd/fetal-health-classification) [Accessed 10 April 2021].

2. Matplotlib.org. n.d. Matplotlib: Python plotting — Matplotlib 3.4.1 documentation. [online] Available at: https://matplotlib.org/ (https://matplotlib.org/) [Accessed 10 April 2021].

3. Seaborn.pydata.org. n.d. seaborn: statistical data visualization — seaborn 0.11.1 documentation. [online] Available at: https://seaborn.pydata.org/index.html (https://seaborn.pydata.org/index.html) [Accessed 10 April 2021]

4. Pandas.pydata.org. n.d. pandas - Python Data Analysis Library. [online] Available at: https://pandas.pydata.org/ (https://pandas.pydata.org/) [Accessed 10 April 2021].

5. Aksakalli, V., Yenice, Z., Wong, Y., Ture, I. and Malekipirbazari, M., n.d. SK Part 5: Advanced Topics: Pipelines, Statistical Model Comparison, and Model Deployment | www.featureranking.com. [online] www.featureranking.com. Available at: https://www.featureranking.com/tutorials/machine-learning-tutorials/sk-part-5-advanced-topics-pipelines-statistical-model-comparison-and-model-deployment/#4 (https://www.featureranking.com/tutorials/machine-learning-tutorials/sk-part-5-advanced-topics-pipelines-statistical-model-comparison-and-model-deployment/#4) [Accessed 29 May 2021].

6. Bade, S., 2019. Is K = 1 is good for KNN?. [online] stats.stackexchange.com. Available at: https://stats.stackexchange.com/questions/440064/is-k-1-is-good-for-knn-when-error-is-min-accuracy-is-max-and-even-auroc-is-m (https://stats.stackexchange.com/questions/440064/is-k-1-is-good-for-knn-when-error-is-min-accuracy-is-max-and-even-auroc-is-m) [Accessed 29 May 2021].

7. Brownlee, J., 2016. Overfitting and Underfitting With Machine Learning Algorithms. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/ (https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/) [Accessed 29 May 2021].

8. Donges, N., 2019. A complete guide to the random forest algorithm. [online] BuiltIn. Available at: https://builtin.com/data-science/random-forest-algorithm (https://builtin.com/data-science/random-forest-algorithm) [Accessed 29 May 2021].

9. en.wikipedia.org. n.d. List of countries by maternal mortality ratio. [online] Available at: https://en.wikipedia.org/wiki/List_of_countries_by_maternal_mortality_ratio (https://en.wikipedia.org/wiki/List_of_countries_by_maternal_mortality_ratio) [Accessed 29 May 2021].

10. Brownlee, J., 2019. Tune Hyperparameters for Classification Machine Learning Algorithms. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/ (https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/) [Accessed 29 May 2021].

11. Scikit-learn.org. n.d. scikit-learn: machine learning in Python — scikit-learn 0.24.2 documentation. [online] Available at: https://scikit-learn.org/stable/ (https://scikit-learn.org/stable/) [Accessed 29 May 2021].