BM59D Project Report

Berna Erden

December 20, 2017

1 The Definition of Problem

Cardiotocography (CTG) is a technical means of recording the fetal heartbeat and the uterine contractions during pregnancy. Cardiotocography may inform the state of fetal to be normal, suspicious or pathologic. In this project, we will predict the fetal state each patient on provided cardiotocograms dataset. The dataset consists of measurements of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms classified by expert obstetricians. The size of dataset is 2126 fetal cardiotocograms (CTGs), and it includes automatically processed diagnostic features. The proposed algorithm will classify the pattern of a fetal state (Normal, Suspect, Pathologic). The following features are provided in the dataset:

1. LB - FHR baseline (beats per minute)
2. AC - number of accelerations per second
3. FM - number of fetal movements per second
4. UC - number of uterine contractions per second
5. DL - number of light decelerations per second
6. DS - number of severe decelerations per second
7. DP - number of prolongued decelerations per second
8. ASTV - percentage of time with abnormal short term variability
9. MSTV - mean value of short term variability
10. ALTV - percentage of time with abnormal long term variability
11. MLTV - mean value of long term variability
12. Width - width of FHR histogram
13. Min - minimum of FHR histogram
14. Max - Maximum of FHR histogram
15. Nmax - number of histogram peaks
16. Nzeros - number of histogram zeros

17. Mode - histogram mode
18. Mean - histogram mean
19. Median - histogram median
20. Variance - histogram variance
21. Tendency - histogram tendency
22. CLASS - FHR pattern class code (1 to 10) (not be predicted in this case)
23. NSP - fetal state class code (N=normal; S=suspect; P=pathologic) (predicted label)

2 Dataset Analysis

The provided dataset is divided into two sets: training and test set. The training set and test set consist of 1701 and 425 instances respectively. The following analyses are made on the training set.

2.1 The Histograms

The Figure 1 shows the distribution of each feature in the dataset. The half of the features are bounded on one peak or two peaks. If we look at the other features, they do not fit a specific distribution like Gaussian except of the features related to FHR histogram.

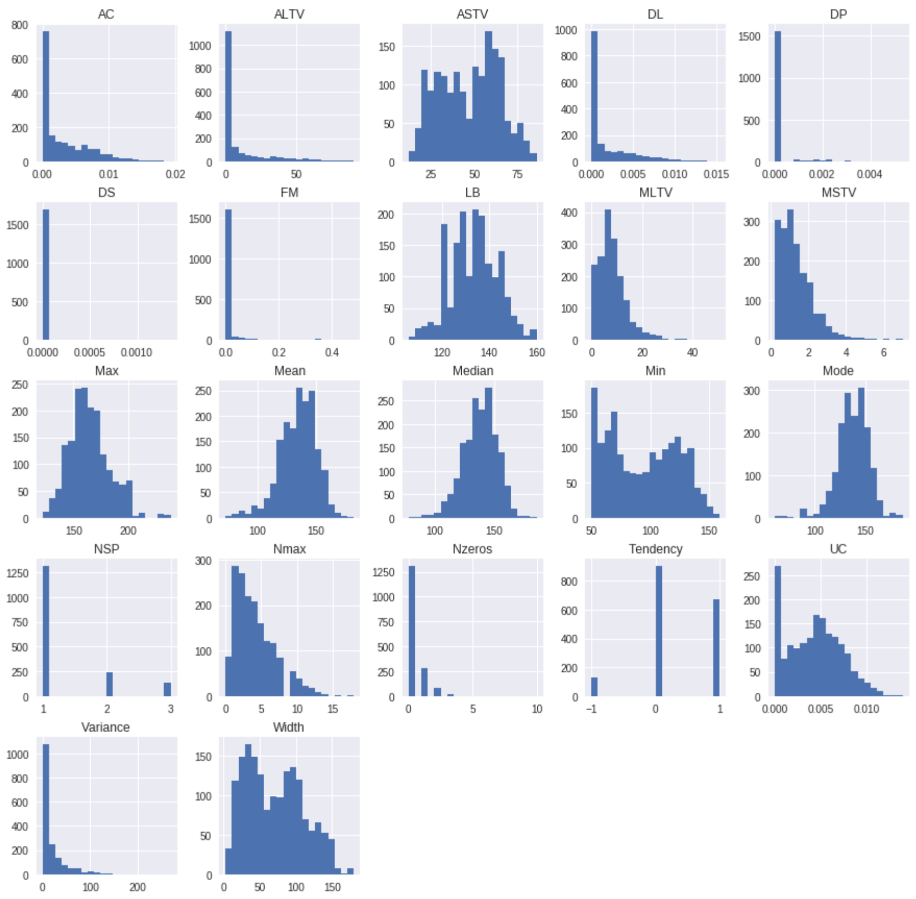


Figure 1: The Histograms

2.2 The Correlation Analysis

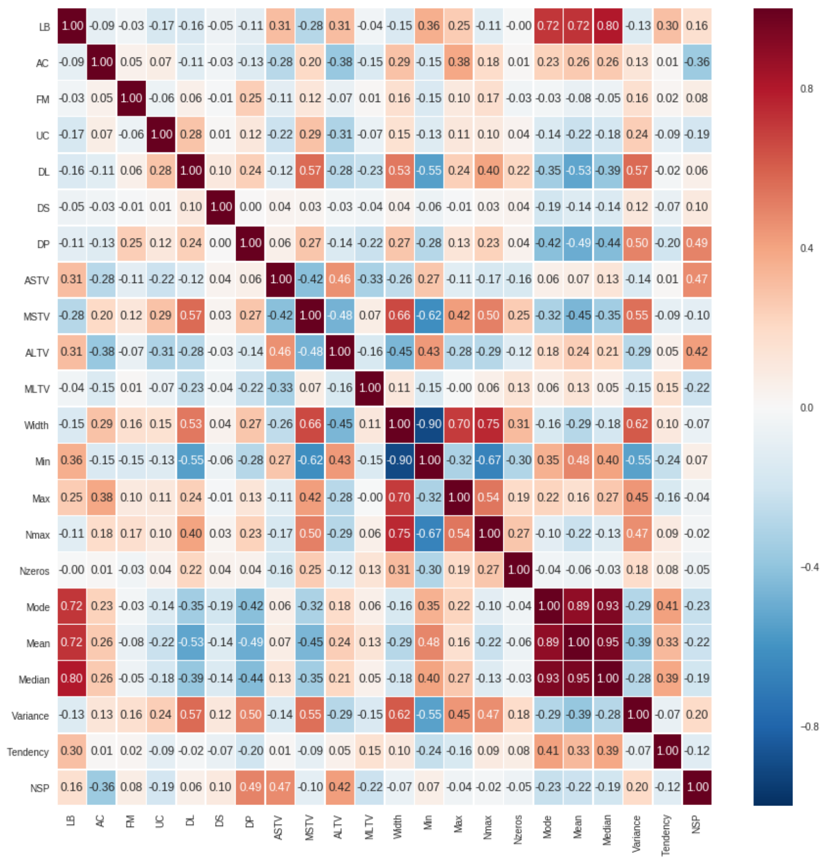


Figure 2: The Correlation

The features have low correlation with the target variable, which indicates that there is a weak linear relationship with the target. On the one hand, the most correlated four features show that the data belonging to Normal and Suspect classes are mostly collected on the same area, which indicates that they are not separable. The features ASTV and ALTV are equally distributed in each class data. On the other hand, the least correlated four features do not separate boundaries between different classes as well as the most correlated ones.

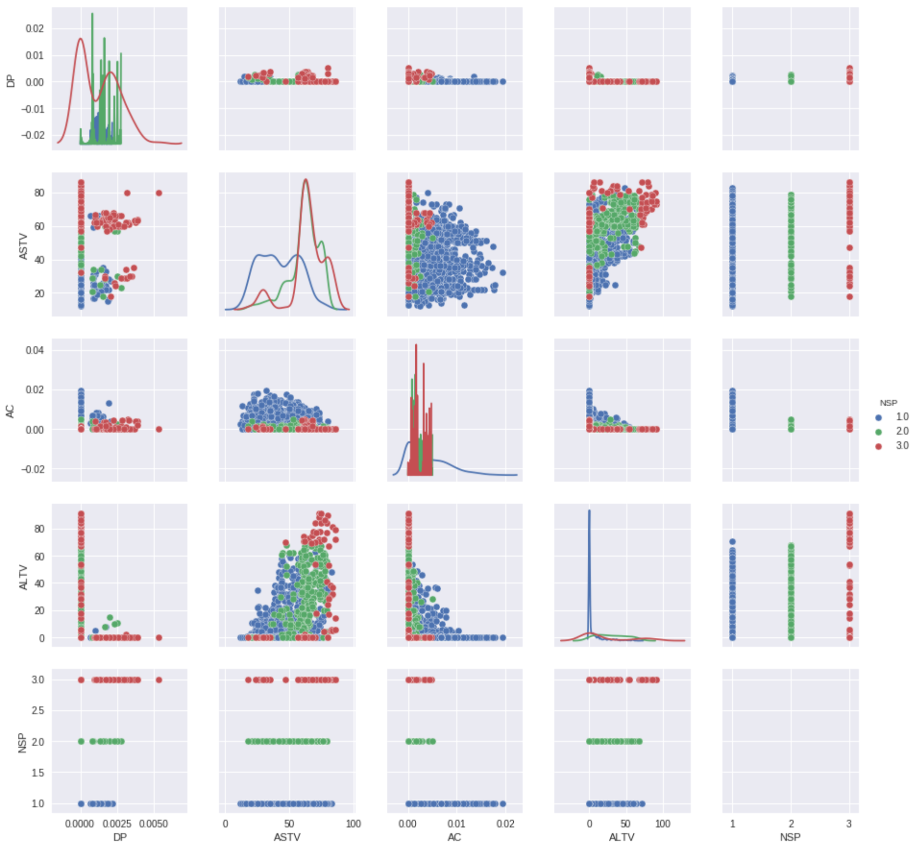


Figure 3: The Most Correlated Features

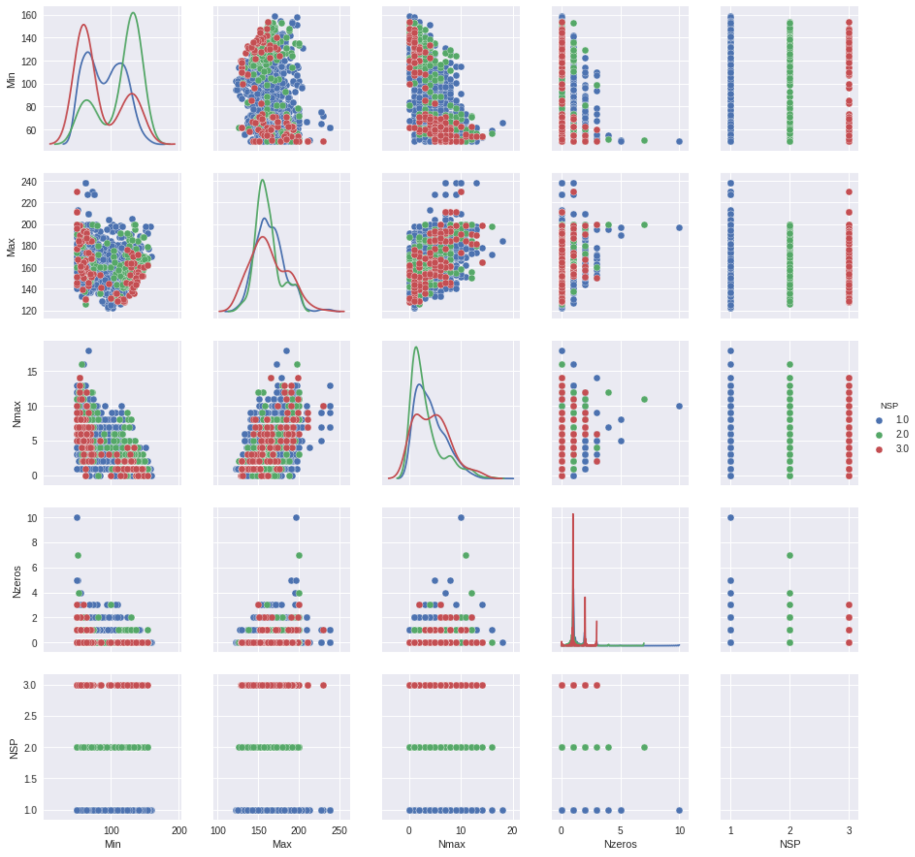


Figure 4: The Least Correlated Features

3 Model Development

In this project, we propose a solution which implements an ensemble learning model to the multiclass classification problem in order to boost performance. Firstly, we choose the three learning algorithms to be implemented within ensemble classifier: Logistic Regression, Support Vector Machine and Xgboost.

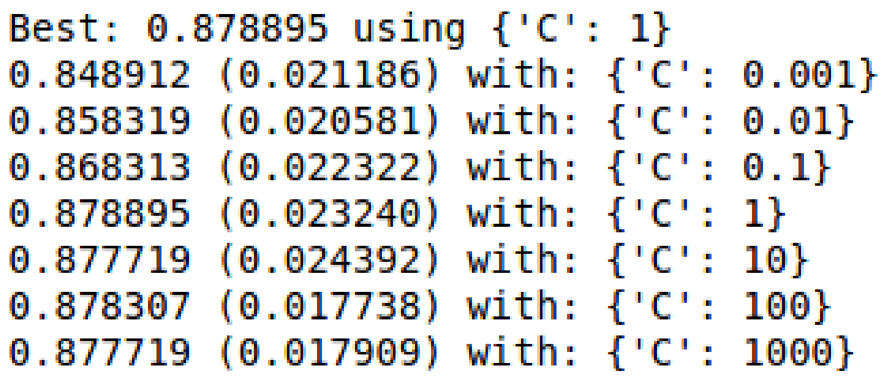
Logistic regression measures the relationship between the categorical dependent variable and the independent variables by using logistic function which is calculated by class probabilities. It is easy to implement and performs well on linearly separable classes.

SVM is an machine learning algorithm whose objective is to maximize the margin which is the distance between the hyperplane (decision boundary) and the training samples that are closest to this hyperplane. SVM with kernel trick allows to project the combinations of the original features onto a higher dimensional space where it becomes linearly separable.

Xgboost, "Extreme Gradient Boosting", is an advanced implementation of gradient boosted decision trees designed for speed and performance. It performs additive optimization in functional space instead of using traditional optimization methods in Euclidean space [1]. Xgboost performs well in high dimensional spaces as well as large number of training examples.

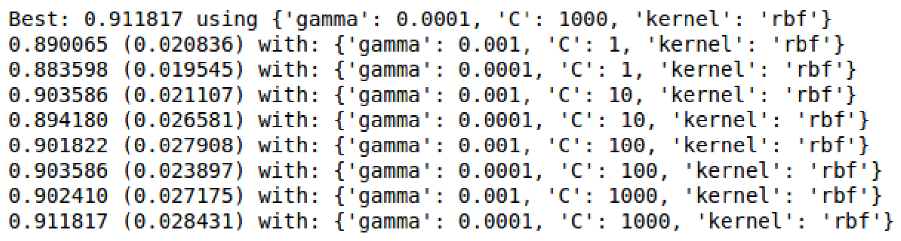
Cross validation is the process of training the model using one subset of data and validating it using a different subset of data in order to estimate the generalization performance of the model. Parameter tuning is to select the optimal hyperparameters to further improve the performance for making predictions on test data.

To calculate the average performance of the chosen models and to tune hyperparameters of them , we implement grid search cross validation using Stratified-K-Fold (10-fold). The following results are the training accuracies and best parameters of respectively Logistic Regression, SVM and Xgboost:



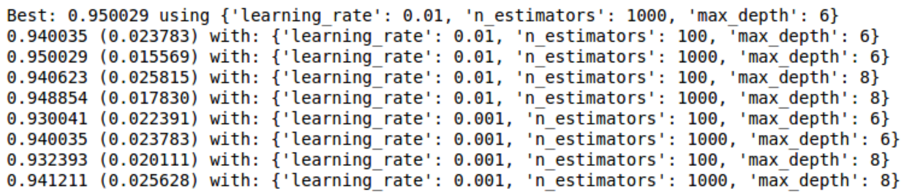
```
Best: 0.878895 using {'C': 1}
0.848912 (0.021186) with: {'C': 0.001}
0.858319 (0.020581) with: {'C': 0.01}
0.868313 (0.022322) with: {'C': 0.1}
0.878895 (0.023240) with: {'C': 1}
0.877719 (0.024392) with: {'C': 10}
0.878307 (0.017738) with: {'C': 100}
0.877719 (0.017909) with: {'C': 1000}
```

Figure 5: Logistic Regression Grid Cv Results



```
Best: 0.911817 using {'gamma': 0.0001, 'C': 1000, 'kernel': 'rbf'}
0.890065 (0.020836) with: {'gamma': 0.001, 'C': 1, 'kernel': 'rbf'}
0.883598 (0.019545) with: {'gamma': 0.0001, 'C': 1, 'kernel': 'rbf'}
0.903586 (0.021107) with: {'gamma': 0.001, 'C': 10, 'kernel': 'rbf'}
0.894180 (0.026581) with: {'gamma': 0.0001, 'C': 10, 'kernel': 'rbf'}
0.901822 (0.027908) with: {'gamma': 0.001, 'C': 100, 'kernel': 'rbf'}
0.903586 (0.023897) with: {'gamma': 0.0001, 'C': 100, 'kernel': 'rbf'}
0.902410 (0.027175) with: {'gamma': 0.001, 'C': 1000, 'kernel': 'rbf'}
0.911817 (0.028431) with: {'gamma': 0.0001, 'C': 1000, 'kernel': 'rbf'}
```

Figure 6: SVM Grid Cv Results



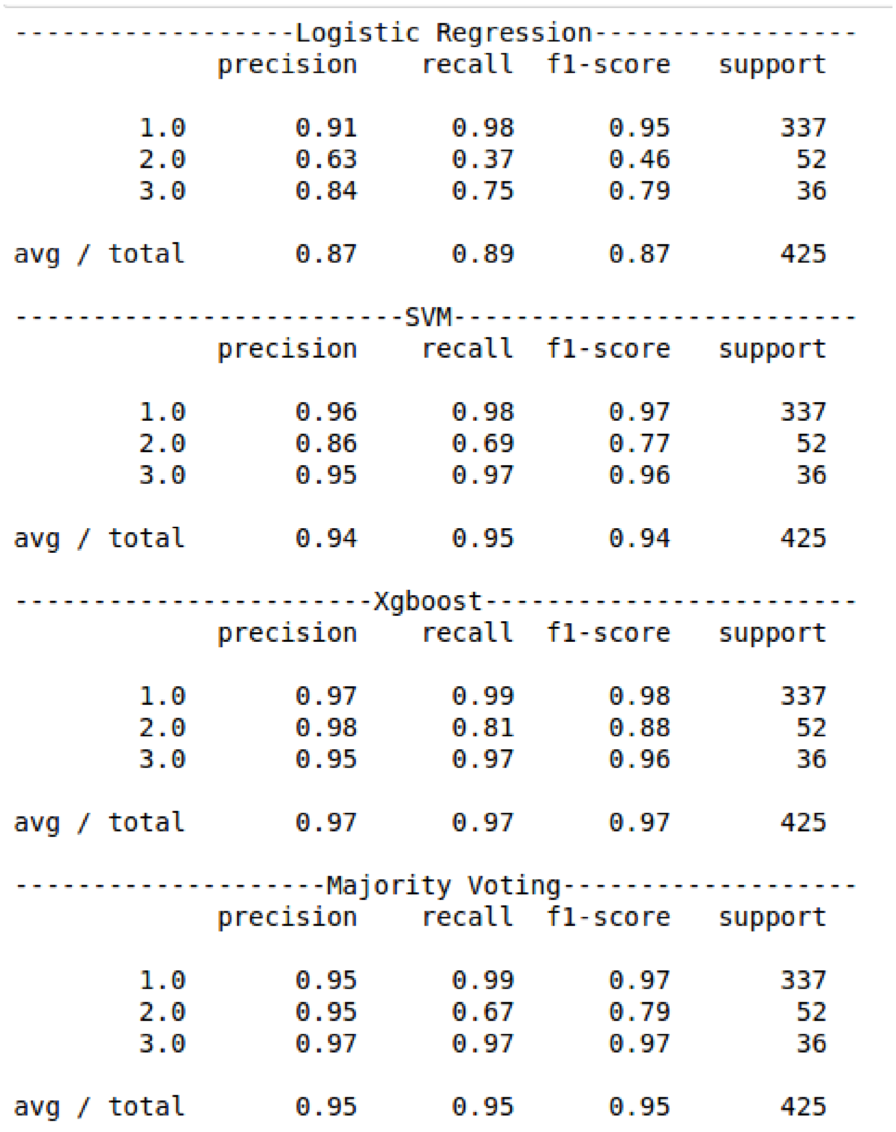
```
Best: 0.950029 using {'learning_rate': 0.01, 'n_estimators': 1000, 'max_depth': 6}
0.940035 (0.023783) with: {'learning_rate': 0.01, 'n_estimators': 100, 'max_depth': 6}
0.950029 (0.015569) with: {'learning_rate': 0.01, 'n_estimators': 1000, 'max_depth': 6}
0.940623 (0.025815) with: {'learning_rate': 0.01, 'n_estimators': 100, 'max_depth': 8}
0.948854 (0.017830) with: {'learning_rate': 0.01, 'n_estimators': 1000, 'max_depth': 8}
0.930041 (0.022391) with: {'learning_rate': 0.001, 'n_estimators': 100, 'max_depth': 6}
0.940035 (0.023783) with: {'learning_rate': 0.001, 'n_estimators': 1000, 'max_depth': 6}
0.932393 (0.020111) with: {'learning_rate': 0.001, 'n_estimators': 100, 'max_depth': 8}
0.941211 (0.025628) with: {'learning_rate': 0.001, 'n_estimators': 1000, 'max_depth': 8}
```

Figure 7: Xgboost Grid Cv Results

In addition to the three learning algorithms, we develop an ensemble learning algorithm, ”plurality voting”, which combines different classifiers into a meta-classifier to balance out the individual classifiers’ weaknesses on a particular dataset. Plurality voting algorithm selects the class label receiving the most votes from the classifiers. Our ensemble model combines the results of Logistic Regression, SVM and Xgboost.

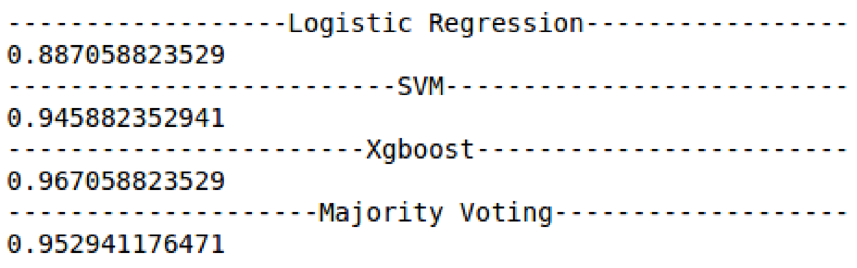
At the last step, we train each model with the best parameters and make predictions on the test set.

4 Evaluation



```
------------------Logistic Regression-----------------
             precision    recall  f1-score   support

        1.0       0.91      0.98      0.95       337
        2.0       0.63      0.37      0.46        52
        3.0       0.84      0.75      0.79        36

avg / total       0.87      0.89      0.87       425

-------------------------SVM-------------------------
             precision    recall  f1-score   support

        1.0       0.96      0.98      0.97       337
        2.0       0.86      0.69      0.77        52
        3.0       0.95      0.97      0.96        36

avg / total       0.94      0.95      0.94       425

-----------------------Xgboost-----------------------
             precision    recall  f1-score   support

        1.0       0.97      0.99      0.98       337
        2.0       0.98      0.81      0.88        52
        3.0       0.95      0.97      0.96        36

avg / total       0.97      0.97      0.97       425

--------------------Majority Voting------------------
             precision    recall  f1-score   support

        1.0       0.95      0.99      0.97       337
        2.0       0.95      0.67      0.79        52
        3.0       0.97      0.97      0.97        36

avg / total       0.95      0.95      0.95       425
```

Figure 8: The Classification Performance



```
------------------Logistic Regression----------------
0.887058823529
-------------------------SVM-------------------------
0.945882352941
-----------------------Xgboost-----------------------
0.967058823529
-------------------Majority Voting-------------------
0.952941176471
```

Figure 9: The Classification Accuracy

As seen in the above tables, Xgboost delivered best performance on the CTG dataset. We expected Plurality Voting to improve the accuracy; however, it failed to reach the performance of Xgboost due to being a linear model. Because of the low correlation between the attributes and the target, the nonlinear models Xgboost and SVM really outperformed Logistic Regression.

Sensitivity means that the fraction of people with the disease that the test correctly identifies as positive. Specificity means that the fraction of people without the disease that the test correctly identifies as negative. ROC curves demonstrates the tradeoff between sensitivity and specificity. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test. As with following ROC curves, Xgboost and SVM were successful at the prediction of Pathologic cases with the accuracy 98%. All classifiers failed at the prediction of Suspect cases.

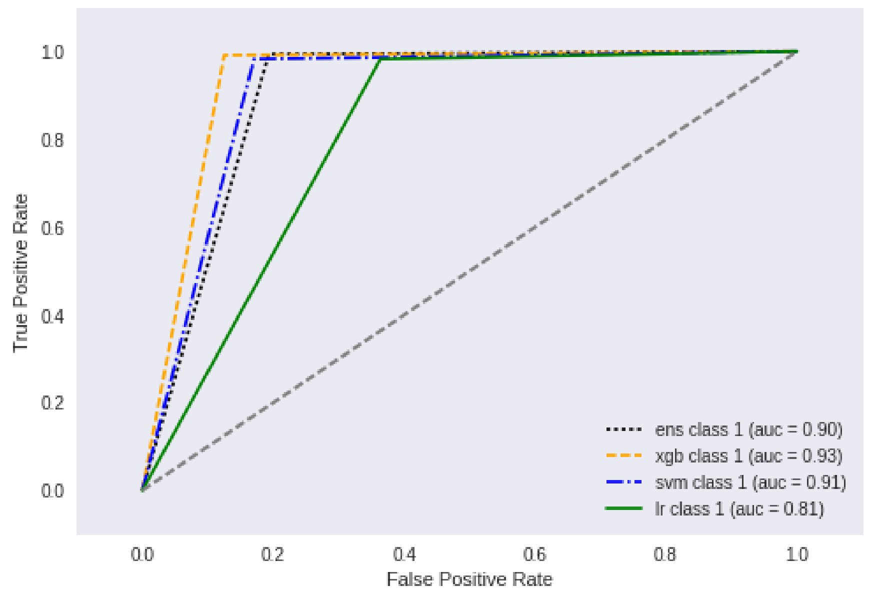


Figure 10: The ROC Curve of Normal Cases

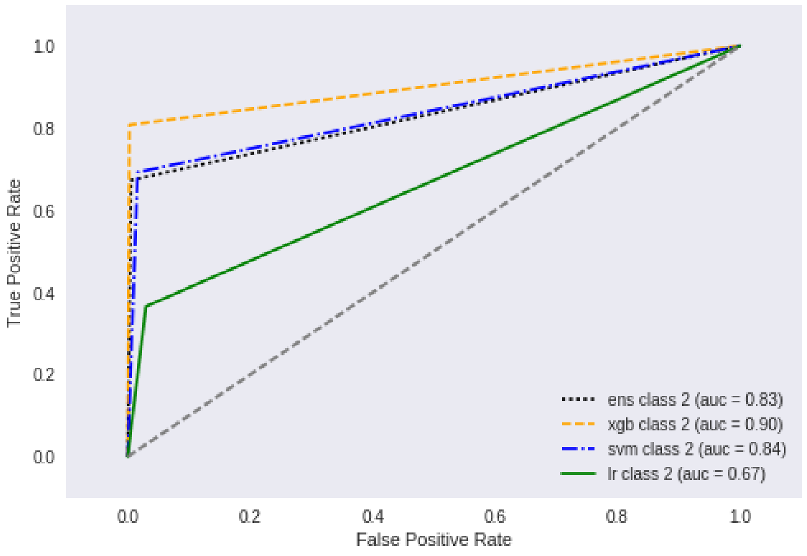


Figure 11: The ROC Curve of Suspect Cases

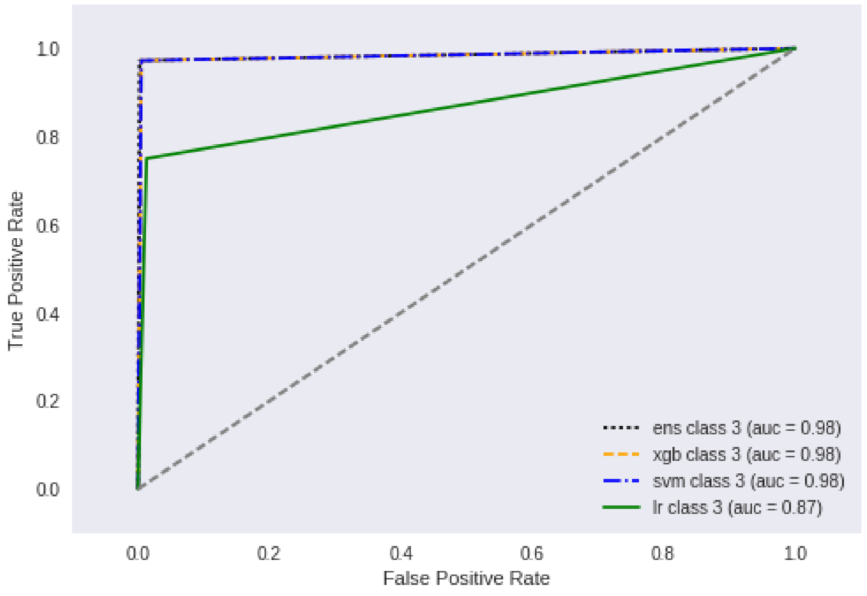


Figure 12: The ROC Curve of Pathologic Cases

5 Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score, roc_curve,auc,
    classification_report
from sklearn.preprocessing import label_binarize
sns.set(color_codes=True)

def display_conf_mat(conf_mat):
      fig, ax = plt.subplots(figsize=(2.5, 2.5))
      ax.matshow(conf_mat, cmap=plt.cm.Blues, alpha=0.3)
      for i in range(conf_mat.shape[0]):
                for j in range(conf_mat.shape[1]):
                        ax.text(x=j, y=i,s=conf_mat[i, j],va='
                            center', ha='center')
      plt.xlabel('predicted label')
      plt.ylabel('true label')
      plt.show()

def grid_search_cv(clf, train, label, param_grid, scoring='accuracy
    '):
      kfold = StratifiedKFold(n_splits=10, shuffle=True,
          random_state=7)
      grid_search = GridSearchCV(clf, param_grid, scoring=scoring,
          n_jobs=-1, cv=kfold)
      grid_result = grid_search.fit(train, label)
      print("Best: %f using %s" % (grid_result.best_score_,
          grid_result.best_params_))
      means = grid_result.cv_results_['mean_test_score']
      stds = grid_result.cv_results_['std_test_score']
      params = grid_result.cv_results_['params']
      for mean, stdev, param in zip(means, stds, params):
        print("%f (%f) with: %r" % (mean, stdev, param))
      return grid_result.best_params_

train = pd.read_csv('train.csv').dropna()
test = pd.read_csv('test.csv').dropna()
features=['LB', 'AC', 'FM', 'UC', 'DL', 'DS', 'DP',
          'ASTV', 'MSTV', 'ALTV', 'MLTV', 'Width',
          'Min', 'Max','Nmax', 'Nzeros', 'Mode',
          'Mean', 'Median', 'Variance', 'Tendency']
target='NSP'

train_array, label_array = np.asarray(train[features]), np.asarray(
    train[target])
test_array, test_label_array = np.asarray(test[features]), np.
    asarray(test[target])
```

```
train.hist(alpha=1.0, bins=20,figsize=(15, 15))
fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(train.corr(),annot=True, fmt=".2f", linewidths=.5,
                            ax=ax)
sns.pairplot(train[['DP', 'ASTV', 'AC', 'ALTV','NSP']],
                                  hue='NSP', diag_kind="kde")
sns.pairplot(train[['Min', 'Max','Nmax', 'Nzeros','NSP']],
                                  hue='NSP', diag_kind="kde")
```

```
xgb_clf = xgb.XGBClassifier(objective='multi:softmax', nthread=4)
xgb_param_grid = dict(n_estimators=[100, 1000],
                                  learning_rate=[0.01, 0.001],
                 max_depth=[6,8])
xgb_best_params = grid_search_cv(xgb_clf,train_array,label_array ,
    xgb_param_grid)
xgb_clf.set_params(n_estimators= xgb_best_params['n_estimators'],
                  learning_rate= xgb_best_params['learning_rate'],
                  max_depth= xgb_best_params['max_depth'])
xgb_clf.fit(train[features], train[target],eval_metric='auc',
    verbose=True)
test['XGB_Pred'] = xgb_clf.predict(test[features])
xgb_acc = accuracy_score(test[target], test['XGB_Pred'])
print(xgb_acc)
feat_imp = pd.Series(xgb_clf.booster().get_fscore()).sort_values(
    ascending=False)
feat_imp.plot(kind='bar', title='Feature Importances')
plt.ylabel('Feature Importance Score')

svm_clf = SVC(decision_function_shape='ovr')
svm_param_grid = [{'C': [1,10, 100, 1000],
                                   'gamma': [0.001, 0.0001],
                    'kernel': ['rbf']},]
svm_best_params = grid_search_cv(svm_clf,train_array, label_array,
    svm_param_grid)
svm_clf.set_params(C= svm_best_params['C'],
gamma= svm_best_params['gamma'],
kernel= svm_best_params['kernel'])
svm_clf.fit(train[features], train[target])
test['SVM_Pred'] = svm_clf.predict(test[features])
svm_acc = accuracy_score(test[target],test['SVM_Pred'] )
print(svm_acc)

lr_clf = LogisticRegression(multi_class='ovr')
lr_param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000] }
lr_best_params = grid_search_cv(lr_clf,train_array, label_array,
    lr_param_grid)
lr_clf.set_params(C= lr_best_params['C'])
lr_clf.fit(train[features], train[target])
test['LR_Pred'] = lr_clf.predict(test[features])
lr_acc = accuracy_score(test[target],test['LR_Pred'] )
print(lr_acc)

ens_clf = VotingClassifier(estimators=[
('xgb', xgb_clf), ('svm', svm_clf), ('lr', lr_clf)], voting='hard')
ens_clf = ens_clf.fit(train[features],train[target])
test['ENS_Pred'] = ens_clf.predict(test[features])
ens_acc = accuracy_score(test[target],test['ENS_Pred'] )
print(ens_acc)
print('--------------------Logistic Regression-------------------')
print(lr_acc)
print('------------------------------SVM-------------------------------')
print(svm_acc)
print('--------------------------Xgboost---------------------------')
print(xgb_acc)
print('----------------------Majority Voting---------------------')
print(ens_acc)

print('--------------------Logistic Regression-------------------')
print(classification_report(test[target],test['LR_Pred'] ))
print('------------------------------SVM-------------------------------')
print(classification_report(test[target],test['SVM_Pred'] ))
print('--------------------------Xgboost---------------------------')
print(classification_report(test[target],test['XGB_Pred'] ))
print('----------------------Majority Voting---------------------')
print(classification_report(test[target],test['ENS_Pred'] ))


target_bin = label_binarize(test[target], classes=[1,2,3])
ens_bin = label_binarize(test['ENS_Pred'], classes=[1,2,3])
xgb_bin = label_binarize(test['XGB_Pred'], classes=[1,2,3])
svm_bin = label_binarize(test['SVM_Pred'], classes=[1,2,3])
lr_bin = label_binarize(test['LR_Pred'], classes=[1,2,3])
colors = ['black', 'orange', 'blue', 'green']
linestyles = [':', '--', '-.', '-']
labels=['ens','xgb','svm','lr']
classifiers = [ens_bin,xgb_bin,svm_bin,lr_bin]
n_classes = 3

for i in range(n_classes):
        for clf, label, clr, ls in zip(classifiers,labels, colors,
            linestyles):
      fpr, tpr, thresholds = roc_curve(target_bin[:,i],clf[:,i])
      roc_auc = auc(x=fpr, y=tpr)
      plt.plot(fpr, tpr,
      color=clr,
      linestyle=ls,
      label='%s class %s (auc = %0.2f)' % (label,i+1, roc_auc))

    plt.legend(loc='lower right')
    plt.plot([0, 1], [0, 1],
    linestyle='--',
    color='gray',
    linewidth=2)
    plt.xlim([-0.1, 1.1])
    plt.ylim([-0.1, 1.1])
    plt.grid()
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.show()
```

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.