Final Project

John Raphael Fox

Arizona State University

Abstract

In this paper, we will present the results of an exhaustive study of a dataset given by the instructor of the course. Preprocessing of the data was done, where we dealt with: standardization of the continuous variables and encoding of categorical variables. We also fit a variety of different models to the data; which included different SVMs, Neural Networks, and Decision Trees.

Final Project

The objective of this project is to illustrate what was learnt in class this past semester. We will illustrate the process of fitting an appropriate model to a dataset. We will use the diagnostic tools taught in class, for the better selection of the model. We will finally fit the selected model to a test set from which the results will be turned in. The code that was used will be added to the end of this paper.

Preprocessing

In the data set, different data types were present. Which signified, we had to preprocess them differently. There was a total of 14 features that seemed to be continuous, these features were standardized. The existing categorical variables were also encoded.

Model Selection

After preprocessing the data, we continued by fitting a variety of different models; and varied the values of the parameters involved, via GridSearchCV or RandomizeSearchCV, even manually. We will now list some of the models that were involved in the model selection process:

- Support Vector Machine
  - Linear
  - Radial Base Function
  - Sigmoid
- Neural Networks
  - Used a variety of NN.
- Decision Tree

A variety of different tools were used to check the validity of each model. We now present the list of diagnostics that were used:

- Cross-validation error
- Test Accuracy
- Training Accuracy
- Confusion Matrix
- Precision
- Recall
- F1-score
- ROC Curve
- Learning Curve
- And more.

After analyzing many diagnostics (check code) the conclusion that the best model, of the tested one's, was Support Vector Machine, with $C = 0.1$, $gamma = 0.001$, $kernel = linear$, $random\_state = 123$.

Results

The following are the results of the diagnostics, starting with a $test\ accuracy = 84.22\%$, a $training\ accuracy = 85.23\%$ and a $cross - validation\ Error = 84.32\%$.

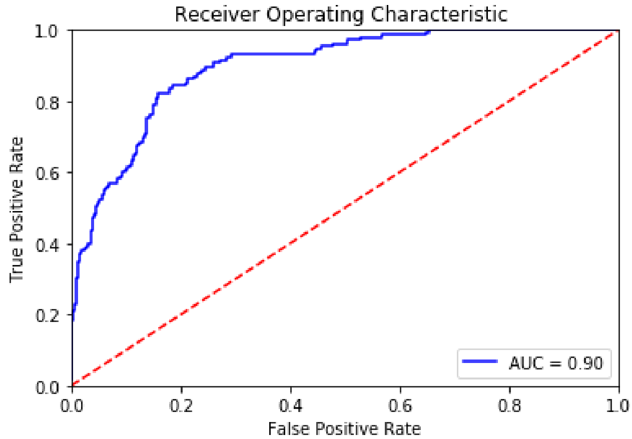
Confusion Matrix

```
Predicted  -1.0  1.0   __all__
Actual
-1.0       2210  150     2360
1.0         340  425      765
__all__    2550  575     3125
```

Precision, Recall, and f1-Score

```
             precision    recall  f1-score   support

       -1.0       0.87      0.94      0.90      2360
        1.0       0.74      0.56      0.63       765

avg / total       0.84      0.84      0.84      3125
```

ROC Curve



Learning Curve

Others

```
TPR: 0.555555555556
TNR: 0.936440677966
PPV: 0.739130434783
NPV: 0.866666666667
FPR: 0.0635593220339
FDR: 0.260869565217
FNR: 0.444444444444
ACC: 0.8432
F1_score: 0.634328358209
```

After choosing a model that seems to be accurate, we were asked to fit it to a test dataset. The results are zipped with this file named BMI555IEE520_Results2018_JohnRaphaelFox.csv.

Conclusion

There are many factors that are involved in the Data Mining Process. These factors include, but are not limited to: data selection, data pre-processing, data mining and data post-processing. We mostly dealt with the second and third, because the data set was given to us. This process enabled us to obtain, what we think, is an appropriate model for the data.

FinalProject

December 4, 2018

0.1 Appendix

```
In [14]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         from sklearn import preprocessing
         from sklearn.preprocessing import LabelBinarizer
         from sklearn import metrics
         from sklearn import model_selection
         from pandas_ml import ConfusionMatrix
```

```
In [115]: from sklearn.model_selection import learning_curve
          def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                                  n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
              """
              Generate a simple plot of the test and training learning curve.

              Parameters
              ----------
              estimator : object type that implements the "fit" and "predict" methods
                  An object of that type which is cloned for each validation.

              title : string
                  Title for the chart.

              X : array-like, shape (n_samples, n_features)
                  Training vector, where n_samples is the number of samples and
                  n_features is the number of features.

              y : array-like, shape (n_samples) or (n_samples, n_features), optional
                  Target relative to X for classification or regression;
                  None for unsupervised learning.

              ylim : tuple, shape (ymin, ymax), optional
                  Defines minimum and maximum yvalues plotted.

              cv : int, cross-validation generator or an iterable, optional
                  Determines the cross-validation splitting strategy.
```

```
        Possible inputs for cv are:
          - None, to use the default 3-fold cross-validation,
          - integer, to specify the number of folds.
          - An object to be used as a cross-validation generator.
          - An iterable yielding train/test splits.

        For integer/None inputs, if ``y`` is binary or multiclass,
        :class:`StratifiedKFold` used. If the estimator is not a classifier
        or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

        Refer :ref:`User Guide <cross_validation>` for the various
        cross-validators that can be used here.

    n_jobs : int or None, optional (default=None)
        Number of jobs to run in parallel.
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
        for more details.

    train_sizes : array-like, shape (n_ticks,), dtype float or int
        Relative or absolute numbers of training examples that will be used to
        generate the learning curve. If the dtype is float, it is regarded as a
        fraction of the maximum size of the training set (that is determined
        by the selected validation method), i.e. it has to be within (0, 1].
        Otherwise it is interpreted as absolute sizes of the training sets.
        Note that for classification the number of samples usually have to
        be big enough to contain at least one sample from each class.
        (default: np.linspace(0.1, 1.0, 5))
    """
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
```

```
        plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Training score")
        plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")

        plt.legend(loc="best")
        return plt
```

In [3]:
```
#load data to pandas dataframe
data = pd.read_csv("train.csv", header=0)
#Because ot the python convention of arrays starting from zero the las row became NaN'
data = data.dropna(axis=0)
#Seperate our regressors and target
X = data.drop(data.columns[67], axis = 1)

X = X.drop(data.columns[0], axis = 1)
y = data.iloc[:,67]
```

In [4]:
```
# Scaling the regression type columns of the data
from sklearn.preprocessing import StandardScaler
vars_to_normalize = pd.DataFrame(X[['x1','x6','x7','x8','x9','x10','x11','x12',
                                       'x16', 'x29', 'x32','x34','x38','x44']])
scaler = StandardScaler()
scaler.fit(vars_to_normalize)
X_scaled = pd.DataFrame(scaler.transform(vars_to_normalize), columns=['x1','x6','x7','
                                       'x16', 'x29', 'x32','x34','x38','x44'])
```

In [5]:
```
#Encoding categorical variables
#From one column 5 are created
X_cate_5 = pd.DataFrame(X[['x5']])
X_cate_5 = pd.DataFrame(LabelBinarizer().fit_transform(X_cate_5), columns=['x67','x68'
#From one column 3 are created
X_cate_13 = pd.DataFrame(X[['x13']])
X_cate_13 = pd.DataFrame(LabelBinarizer().fit_transform(X_cate_13),columns=['x72','x73
#From one column 4 are created
X_cate_64 = pd.DataFrame(X[['x64']])
X_cate_64 = pd.DataFrame(LabelBinarizer().fit_transform(X_cate_64), columns=['x75','x7
#From one column 4 are created
X_cate_65 = pd.DataFrame(X[['x65']])
X_cate_65 = pd.DataFrame(LabelBinarizer().fit_transform(X_cate_65), columns=['x79','x8

X_categorical = pd.concat([X_cate_5, X_cate_13, X_cate_64, X_cate_65], axis=1)
```

In [6]:
```
#Putting all columns together
X_without_normalized_and_categorical = X.drop(['x1','x5', 'x6', 'x7','x8','x9','x10','
                             'x12', 'x13','x64','x65','x16', 'x29', 'x32','x34','x38','x4
X_final = pd.concat([X_scaled, X_without_normalized_and_categorical,X_categorical], ax
```

In [ ]:
```
print(X_final)
```

```
In [8]: # split X and y into training and 25% (default) testing sets
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X_final, y, random_state=0)
```

0.2 Trying out SVM with linear, rbf, and sigmoid kernels

```
In [9]: #Grid search linear, rbf and sigmoid models
        from sklearn.model_selection import GridSearchCV
        from sklearn.svm import SVC


        svc = SVC(probability=True)
        svc_grid = GridSearchCV(svc, {'kernel':['linear','rbf','sigmoid'],'random_state':[123]
                                      'C':[0.01,0.1,1,10]},return_train_score=True)
        svc_grid.fit(X_train, y_train)
        svc_best=svc_grid.best_estimator_

        print ('Best parameters are:',svc_grid.best_params_)
```

```
Best parameters are: {'C': 0.1, 'gamma': 0.001, 'kernel': 'linear', 'random_state': 123}
```

```
In [10]: #Check test accuracy of above "best" Support Vector Machine model
         print ("The Test Accuracy is",np.round(svc_best.score(X_test, y_test)*100,2),"%")
         #Check training accuracy above Support Vector Machine model
         print ("The Training Accuracy is",np.round(svc_best.score(X_train, y_train)*100,2),"%'
```

```
The Test Accuracy is 84.32 %
The Training Accuracy is 85.23 %
```

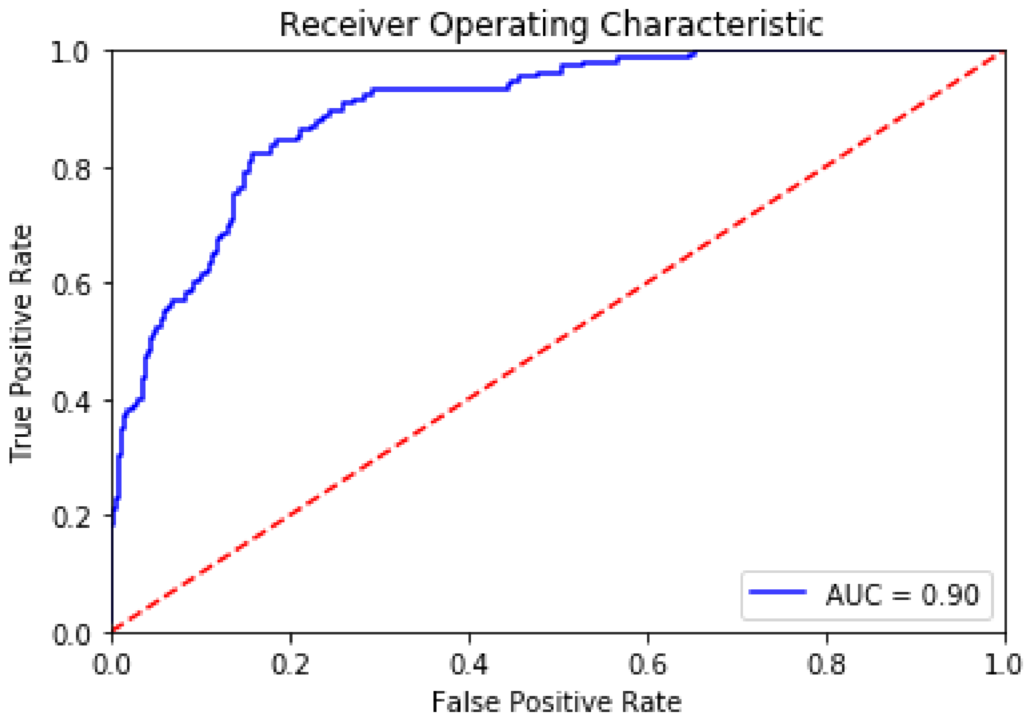
```
In [ ]: from sklearn.metrics import classification_report
        from sklearn.metrics import confusion_matrix

        y_pred = svc_best.predict(X_test)
        print("Testing Error")
        print (metrics.accuracy_score(y_test, y_pred))
        print (classification_report(y_test,y_pred))


        print ("The Confusion Matrix looks like this: \n", confusion_matrix(y_test, y_pred))

        cm = ConfusionMatrix(y_test, y_pred)
        print (cm)
        cm.print_stats()
```

```
In [16]: seed=719
         actuals=[]
         probs=[]
```

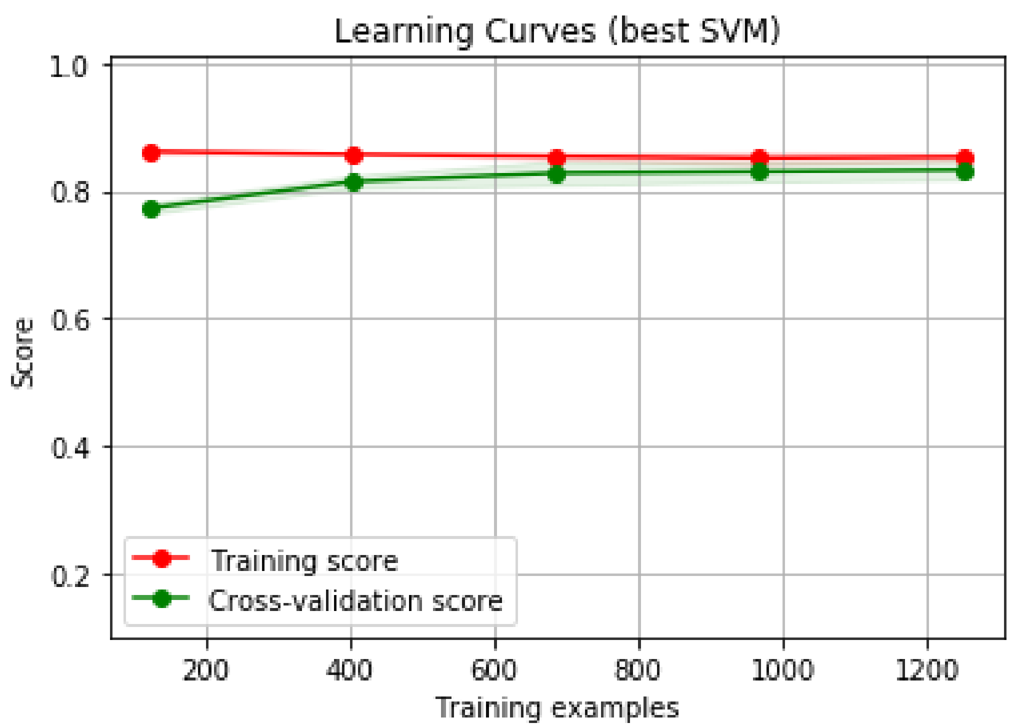
```
hats=[]

kfold = model_selection.KFold(n_splits=5, shuffle=True, random_state=seed)
for train, test in kfold.split(X_final, y):
    #print('train: %s, test: %s' % (train, test))
    # Train classifier on training data, predict test data
    svc_best.fit(X_train, y_train)
    foldhats = svc_best.predict(X_test)
    foldprobs = svc_best.predict_proba(X_test)[:,1] # Class probability estimates for
    actuals = np.append(actuals, y_test) #Combine targets, then probs, and then predi
    probs = np.append(probs, foldprobs)
    hats = np.append(hats, foldhats)
```

In [ ]:
```
print ("Crossvalidation Error")
print ("CVerror = ", metrics.accuracy_score(actuals,hats))
print (metrics.classification_report(actuals, hats))
cm = ConfusionMatrix(actuals,hats)
print (cm)
cm.print_stats()
```

In [18]:
```
#ROC Curve
fpr, tpr, threshold = metrics.roc_curve(actuals, probs)
roc_auc = metrics.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic ')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




```
In [116]: plot_learning_curve(svc_best, "Learning Curves (best SVM)", X_train, y_train, ylim=((
          plt.show()
```



0.3 Neural Networks

In [ ]:
```
from sklearn.neural_network import MLPClassifier
#Neural network model
modelnow=MLPClassifier(hidden_layer_sizes=(5,2,5), random_state=0)
modelnow.fit(X_final,y)
```

In [ ]:
```
# Compute training error
yhat = modelnow.predict(X_train)
print ("Training Error")
print (metrics.accuracy_score(y_train, yhat))
print (metrics.classification_report(y_train, yhat))
print (ConfusionMatrix(y_train, yhat))
```

In [111]:
```
#Cross-validation
seed=719
actuals=[]
probs=[]
hats=[]

kfold = model_selection.KFold(n_splits=5, shuffle=True, random_state=seed)
for train, test in kfold.split(X_final, y):
    #print('train: %s, test: %s' % (train, test))
    # Train classifier on training data, predict test data
    modelnow.fit(X_train, y_train)
    foldhats = modelnow.predict(X_test)
    foldprobs = modelnow.predict_proba(X_test)[:,1] # Class probability estimates fo
    actuals = np.append(actuals, y_test) #Combine targets, then probs, and then pred
    probs = np.append(probs, foldprobs)
    hats = np.append(hats, foldhats)
```

In [ ]:
```
print ("Crossvalidation Error")
print ("CVerror = ", metrics.accuracy_score(actuals,hats))
print (metrics.classification_report(actuals, hats))
cm = ConfusionMatrix(actuals,hats)
print (cm)
cm.print_stats()
```

In [ ]:
```
#ROC Curve
fpr, tpr, threshold = metrics.roc_curve(actuals, probs)
roc_auc = metrics.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic ')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
```

```
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

In [ ]:
```
#Grid search
mlp = MLPClassifier()                #hidden_layer_sizes=(10,15,5), random_state=0)
mlp_grid = GridSearchCV(mlp, {'hidden_layer_sizes':[(5,5,5),(5,6,7),(5,3),(5,10,2),(8,
                        return_train_score=True)
mlp_grid.fit(X_train, y_train)
mlp_best=mlp_grid.best_estimator_

#print ('Best parameters are:',svc_grid.best_params_)
```

In [ ]:
```
print ('Best parameters are:',mlp_grid.best_params_)
```

In [ ]:
```
seed=719
actuals=[]
probs=[]
hats=[]

kfold = model_selection.KFold(n_splits=5, shuffle=True, random_state=seed)
for train, test in kfold.split(X_final, y):
    #print('train: %s, test: %s' % (train, test))
    # Train classifier on training data, predict test data
    mlp_best.fit(X_train, y_train)
    foldhats = mlp_best.predict(X_test)
    foldprobs = mlp_best.predict_proba(X_test)[:,1] # Class probability estimates for
    actuals = np.append(actuals, y_test) #Combine targets, then probs, and then predic
    probs = np.append(probs, foldprobs)
    hats = np.append(hats, foldhats)
```

In [ ]:
```
print ("Crossvalidation Error")
print ("CVerror = ", metrics.accuracy_score(actuals,hats))
print (metrics.classification_report(actuals, hats))
cm = ConfusionMatrix(actuals,hats)
print (cm)
cm.print_stats()
```

In [ ]:
```
# Compute training error
yhat = mlp_best.predict(X_train)
print ("Training Error")
print (metrics.accuracy_score(y_train, yhat))
print (metrics.classification_report(y_train, yhat))
print (ConfusionMatrix(y_train, yhat))
```

In [ ]:
```
#ROC Curve
fpr, tpr, threshold = metrics.roc_curve(actuals, probs)
roc_auc = metrics.auc(fpr, tpr)
```

```
plt.title('Receiver Operating Characteristic ')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

0.4 Decision tree

```
In [24]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.model_selection import RandomizedSearchCV
```

```
In [ ]: #A specific DecisionTreeClassifier later on we will use grid search to find "best" mod
        modelnow = DecisionTreeClassifier(criterion="gini", splitter="best", max_depth=None, mi
                                          min_samples_leaf=10, min_weight_fraction_leaf=0.0, ma
                                          random_state=123, max_leaf_nodes=None, min_impurity_d
                                          min_impurity_split=None, class_weight=None, presort=F
        #fit with training data
        modelnow.fit(X_train,y_train)
```

```
In [26]: # make class predictions for the testing set
         y_pred_class = modelnow.predict(X_test)
```

```
In [ ]: y_pred_class.shape
```

```
In [ ]: print('Accuracy')
        print(metrics.accuracy_score(y_test, y_pred_class))

        print(metrics.classification_report(y_test, y_pred_class))
        print(metrics.confusion_matrix(y_test, y_pred_class))
        cm = ConfusionMatrix(y_test, y_pred_class)
        print (cm)
        cm.print_stats()
```

```
In [ ]: decision_tree_classifier = DecisionTreeClassifier()

        parameter_grid = {'criterion':["gini", "entropy"],'max_depth': [1, 2, 3, 4, 5, 6, 7, 8]
                          'max_features': [1, 2, 3, 4, 5, 6, 7], 'min_samples_split' : [5, 10,
                          'min_samples_leaf':[5,10,15,20,30], 'random_state':[123]}


        grid_search = RandomizedSearchCV(decision_tree_classifier, parameter_grid,cv = 5)

        grid_search.fit(X_train, y_train)

        decision_tree_best=grid_search.best_estimator_
```

```
        print ("Best Score: {}".format(grid_search.best_score_))
        print ("Best params: {}".format(grid_search.best_params_))
```

In [ ]:
```
y_pred_class = decision_tree_best.predict(X_test)
print('Accuracy')
print(metrics.accuracy_score(y_test, y_pred_class))

print(metrics.classification_report(y_test, y_pred_class))
print(metrics.confusion_matrix(y_test, y_pred_class))
cm = ConfusionMatrix(y_test, y_pred_class)
print (cm)
cm.print_stats()
```

In [ ]:
```
#Check test accuracy of above "best" Support Vector Machine model
print ("The Test Accuracy is",np.round(decision_tree_best.score(X_test, y_test)*100,2)
#Check training accuracy above Support Vector Machine model
print ("The Training Accuracy is",np.round(decision_tree_best.score(X_train, y_train)*
```

In [32]:
```
#store the predicted probabilities for class 1
 y_pred_prob = decision_tree_best.predict_proba(X_test)[:, 1]
```

In [ ]:
```
# IMPORTANT: first argument is true values, second argument is predicted probabilities
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
plt.plot(fpr, tpr)
roc_auc = metrics.auc(fpr, tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('ROC curve for  classifier')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.legend(loc = 'lower right')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
plt.show()
```

In [ ]:
```
from sklearn import tree
import graphviz
dot_data = tree.export_graphviz(decision_tree_best, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("TreeModelExample")
graph.view()
```