Bankruptcy Prediction

The task is to predict the likelihood of companies to go bankrupt based on given data of a study over 5 years period.

Bankruptcy Prediction on full data

The task is to predict the bankruptcy in 1, 2 and 3 years. The data contains 5 years of period with horizon of bankruptcy from 5 to 1 year. For this task, only the years 3, 4 and 5 are relevant, as they provide the examples of companies gone bankrupt in 3, 2 and 1 years respectively.

There have been two approaches implemented:

1. 3 individual binary classifiers (each predicting bankrupt or not in 1, 2 and 3 years of horizon)
2. 1 multi-class classifier (predicting not-bankrupt, in 1 year, 2 years or 3 years)

For both approaches, the following models were trained:

- Random Forest
- Logistic Regression
- Multi Layer Perceptron
- Extreme Gradient Boosting

Data Pre-Processing

For both the approaches, the following common data pre-processing tasks were performed:

Imputation

```
# load arff files into memory
raw_data = loadarff(file)
dat = pd.DataFrame(raw_data[0])
# imputation
dat = dat.interpolate()
X = dat.iloc[:, 0:63]
y = dat.iloc[:, 64]
y = y.astype('int')
```

Note that, the data contained several missing values. Several machine learning models require consistency in data and removing observations with missing values would imply loss of significant amount of data. Therefore, for the sake of simplicity and requirement of the modeling, the data have been imputed with interpolation.

Multi Class subject column

For multi-class classification, one step more was done to create a new `mclass` column, that contained the following classes:

- 0 (no bankruptcy)
- 3years (bankruptcy after 3 years)
- 2years (bankruptcy after 2 years)
- 1years (bankruptcy after 1 year)

```
### read arff files and create a custom multi-class column
raw_data = loadarff('data/3year.arff')
dat1 = pd.DataFrame(raw_data[0])
dat1['class'] = dat1['class'].astype('int')
```

```
dat1['mclass'] = np.where(dat1['class'] == 1, '3years', 0)

raw_data = loadarff('data/4year.arff')
dat2 = pd.DataFrame(raw_data[0])
dat2['class'] = dat2['class'].astype('int')
dat2['mclass'] = np.where(dat2['class'] == 1, '2years', 0)

raw_data = loadarff('data/5year.arff')
dat3 = pd.DataFrame(raw_data[0])
dat3['class'] = dat3['class'].astype('int')
dat3['mclass'] = np.where(dat3['class'] == 1, '1years', 0)

### combine all three files into one dataframe
dat = dat1.append(pd.DataFrame(data = dat2), ignore_index=True)
dat = dat.append(pd.DataFrame(data = dat3), ignore_index=True)
```

Random Split

```
# random split
seed = 123
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=seed)
```

The data were split into train and test with the test data having 33% of data. The split was performed randomly and the `random_seed` chosen for the whole task was fixed to `123`.

Binary Classification

```
# main routine to train several models
def main(file, seed=123):
    raw_data = loadarff(file)
    dat = pd.DataFrame(raw_data[0])
    # imputation
    dat = dat.interpolate()
    X = dat.iloc[:, 0:63]
    y = dat.iloc[:, 64]
    y = y.astype('int')
    # random split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=seed)
    # randomForest
    model_rf = RandomForestClassifier(
        n_estimators=1500,
        max_depth=5,
        random_state=seed)
    model_rf.fit(X_train, y_train)
    # LR
    model_lr = LogisticRegression(
        # solver='lbfgs',
        solver='saga',
        multi_class='auto',
        max_iter=1e5,
        tol=1e-4,
        random_state=seed)
    model_lr.fit(X_train, y_train)
    # MLP
    model_mlp = MLPClassifier(
        hidden_layer_sizes=(700, 60),
        activation='relu',
        solver='lbfgs',
        batch_size='auto',
        max_iter=1e7,
        shuffle=True,
        random_state=seed,
        tol=1e-5)
```

```
    model_mlp.fit(X_train, y_train)
    # Xtreme Gradient Boosted
    model_xgb = XGBClassifier(
        max_depth=4,
        n_estimators=1500,
        objective='binary:logistic',
        booster='gbtree',
        n_jobs=-1,
        random_state=seed)
    model_xgb.fit(X_train, y_train)
    # return trained models
    return model_rf,model_lr,model_mlp,model_xgb,X_train,y_train,X_test,y_test



# evaluation of models
def evaluation(y, y_pred):
    cf = confusion_matrix(y, y_pred)
    prec = precision_score(y, y_pred)
    rec = recall_score(y, y_pred)
    f1 = f1_score(y, y_pred)
    auc = roc_auc_score(y, y_pred)
    acc = accuracy_score(y, y_pred)
    return cf, prec, rec, f1, auc, acc
```

Multi-Class Classification

```
### randomForest
model_rf = RandomForestClassifier(
    n_estimators=3000,
    max_depth=3,
    n_jobs=-1,
    random_state=123)
model_rf.fit(X_train, y_train)

### LR
model_lr = LogisticRegression(
    solver='saga',
    multi_class='auto',
    max_iter=1e7,
    tol=1e-5,
    random_state=123)
model_lr.fit(X_train, y_train)

### mlp
model_mlp = MLPClassifier(
    hidden_layer_sizes=(60),
    activation='relu',
    solver='lbfgs',
    batch_size='auto',
    max_iter=1e7,
    shuffle=True,
    random_state=123,
    tol=1e-5)
model_mlp.fit(X_train, y_train)

### xgboost
model_xgb = XGBClassifier(
    max_depth=3,
    n_estimators=3000,
    objective='binary:logistic',
    booster='gbtree',
    n_jobs=-1,
    nthread=None,
```

```
    random_state=123)
model_xgb.fit(X_train, y_train)



### evaluation of models
def evaluation(y, y_pred, mAverage='macro'):
    cf = confusion_matrix(y, y_pred)
    prec = precision_score(y, y_pred, average=mAverage)
    rec = recall_score(y, y_pred, average=mAverage)
    f1 = f1_score(y, y_pred, average=mAverage)
    acc = accuracy_score(y, y_pred)
    return cf, prec, rec, f1, acc
```

Variable Importance and Feature Selection

Feature selection is a typical task to not only select the most important features, but also understand the relations between underlying data.

There exists several feature selection methods such as by evaluating a trained model (e.g. Gini index with Random Forest or Regression Constants with Polynomial Regression); correlation among the features and filtering the highly correlated features.

Here, correlation analysis is done for combined data, in order to see the correlations among the features.

```
# read all .arff files in data folder
arff_files <- list.files(path = 'data', pattern = '.arff', full.names = T)
alldat <- rbindlist(lapply(arff_files[3:5], function(i_file) {
  dat <- read.arff(i_file)
  return(dat)
}))

# impute
thisdat <- na.spline(alldat[, -c(65)])
corr <- cor(thisdat)

# save correlation plot
png('corrplot_impute2.png', width = 12, height = 12, res = 600, units = 'in')
corrplot(corr)
dev.off()
```

Considering only one of the highly correlated columns (more than 0.9), there are 39 features selected.

```
Index(['Attr1', 'Attr2', 'Attr3', 'Attr4', 'Attr5', 'Attr8', 'Attr9', 'Attr12',
       'Attr13', 'Attr15', 'Attr18', 'Attr19', 'Attr20', 'Attr21', 'Attr24',
       'Attr27', 'Attr28', 'Attr29', 'Attr30', 'Attr32', 'Attr33', 'Attr34',
       'Attr36', 'Attr37', 'Attr39', 'Attr40', 'Attr41', 'Attr43', 'Attr45',
       'Attr47', 'Attr55', 'Attr56', 'Attr57', 'Attr58', 'Attr59', 'Attr60',
       'Attr61', 'Attr63', 'Attr64', 'class'],
      dtype='object')
```

Apart from that, feature selection is done by training a random forest for each of the data-file (in R).

```
# read all .arff files in data folder
arff_files <- list.files(path = 'data', pattern = '.arff', full.names = T)

anss <- lapply(arff_files[3:5], function(i_file) {
  dat <- read.arff(i_file)
  # ranomForest for varImp and inital accuracy
```

```
  thisdat <- na.spline(dat[, -c(65)])
  thisdat <- data.table(thisdat, class = factor(dat$class))
  model_rf <- randomForest(factor(class)~., data = thisdat, ntrees = 1500, mtry = 3)
  # importance
  imp <- data.frame(model_rf$importance)
  imp$x <- rownames(imp)
  imp <- data.table(imp)
  imp <- imp[order(-MeanDecreaseGini)]
  imp$x <- factor(imp$x, levels = imp$x)
  return(list(
    model_rf = model_rf
  ))
})


# prepare importance data to visualize
imps <- lapply(anss, function(x) {
  imp <- data.frame(x$model_rf$importance)
  imp$x <- rownames(imp)
  imp <- data.table(imp)
  imp <- imp[order(-MeanDecreaseGini)]
  imp$x <- factor(imp$x, levels = imp$x)
  return(imp)
})

j = 3
for (i in 1:length(imps)) {
  imps[[i]]$year <- paste0('year ', j)
  j <- j+1
}

imps <- rbindlist(imps)

# visualize
require('ggplot2')
plt <- ggplot(imps) + geom_bar(
  aes(x = x, y = MeanDecreaseGini),
  stat = 'identity'
) + theme(
  axis.text.x = element_text(angle = 90)
) + ggtitle(label = 'Variable Importance') + facet_wrap(~year, ncol = 1)

ggsave('varimp.png', plt, width = 12, height = 8, units = 'in', dpi = 600)
```

Further Analysis

- Other imputation methods should be considered, such as
  - clustering of missing values using KNN
  - tree based imputation (missForest)
- Statistical analysis on features to provide more insights about the features for example, using statistical hypothesis testing to find underlying distributions
- Cross Validation with multiple random splits