Ahmed Awadallah

Binary Classification using RandomForest and XGBoost

Introduction

Companies rise and fall. Knowing when they rise and fall can be incredibly lucrative for some interested parties and thus predicting such occurrences has great value. This case study focuses on predicting whether a company will or will not go bankrupt. The dataset contains data on companies across 5 years. The total feature count is 64 with 43405 observations total.

Methods

The dataset is split into 5 different years and thus there are 5 different files to work with. The filetype associated with this data is arff. The data was loaded in, transformed into a dataframe, and then coalesced into one large dataset.

Next the data was explored for general understanding as well as presence of NAs. All the features were found to be continuous so imputation was far easier. There was NAs found all of type np.nan which were handled by feature. If a feature had less than 50% NA then it was imputed using mean imputation otherwise the feature was removed. None of the features had excessive NAs so all were kept and imputed.

Scaling was not performed on the dataset as the algorithms we are using do not suffer from features with different scales.

The target variable was stored as an object datatype so that was converted into integer type as this is expected for some of the models and analysis tools.

The dataset was shuffled to remove any ordering present in the dataset.

Random Forest hyperparameters tuning was accomplished using sklearn random cross validation. Splits of 10 were chosen as this was the number chosen by the paper. Below are the ranges of values searched across.

```
distributions = {
    "criterion": ["gini", "entropy"],
    "max_depth": np.linspace(4, 500, 8).astype(int),
    "min_samples_split": np.linspace(2, 10, 9).astype(int),
    "min_samples_leaf": np.linspace(1, 10, 10).astype(int),
    "min_weight_fraction_leaf": np.logspace(-6, -1, 20),
    "min_impurity_decrease": np.logspace(-6, -1, 20)
}
```

XGBoost hyperparameter tuning was accomplished through a custom function which can be viewed in appendix A. Cross validation splits of 10 were chosen to maintain the same choice as the paper. Below are the range of values searched across.

```
distributions = {
    'eta': np.linspace(0, 1, 16),
    'max_depth': np.linspace(0, 1000, 16).astype(int),
    'min_child_weight': np.logspace(-6, 6, 16),
    'subsample': np.logspace(-6, 0, 8),
    'colsample_bytree': np.logspace(-6, 0, 8),
    'colsample_bylevel': np.logspace(-6, 0, 8),
    'colsample_bynode': np.logspace(-6, 0, 8),
    'lambda': np.logspace(-6, 6, 16),
    'alpha': np.logspace(-6, 6, 16)
}
```

The optimizing metric was auc for both models as this was the metric used in the paper.

Results

AUC Scores RF

```
RF Cross Val AUC
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
auc=0.950493, std=0.008503
```

Figure 1: Random Forest 10 Fold Cross Val
AUC Score and Standard Deviation

Random Forest Achieved an AUC score of 0.95. We can additionally say that we are 95% confident that the AUC lies between 0.944 and 0.956.

Classification Report RF

Random Forest Classification Report

	precision	recall	f1-score	support
0	0.97	1.00	0.98	4136
1	1.00	0.31	0.48	204
accuracy			0.97	4340
macro avg	0.98	0.66	0.73	4340
weighted avg	0.97	0.97	0.96	4340

Figure 2: Random Forest Classification Report for a Single Fold

A value of 0 indicates the company did not go bankrupt while a value of 1 indicates a company did go bankrupt.

Overall the model performs decently. It is important to remember that this dataset is imbalanced as evidenced by the support values. The overall accuracy looks good but we see that specifically predicting if a company goes bankrupt is hard. The precision score is high which shows the model rarely predicts a company bankrupt when it is in fact not bankrupt. The recall score is low which shows the model will often classify bankrupt companies as not bankrupt when they are.

Feature Importance RF

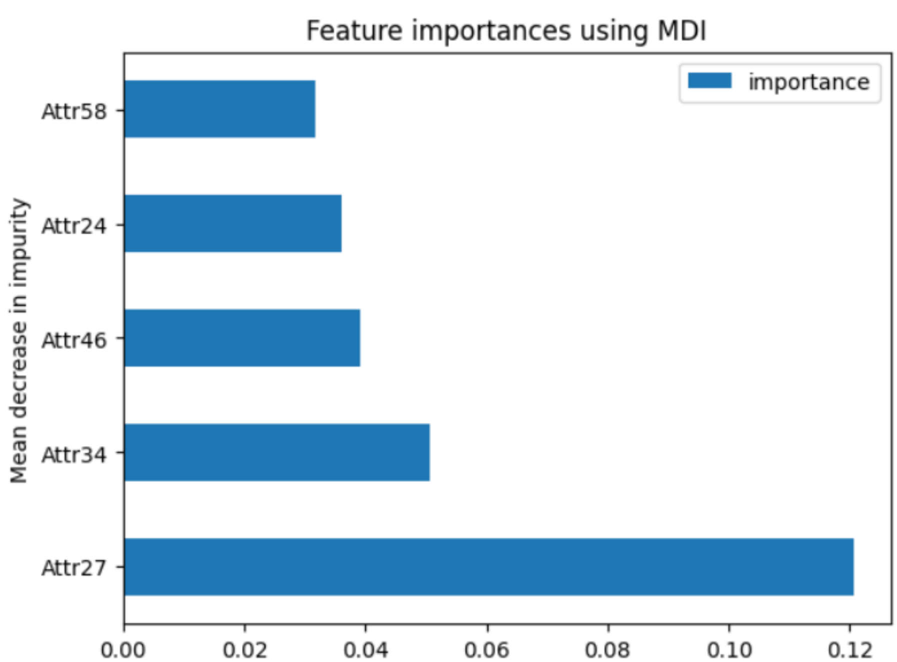

Figure 3: Random Forest Feature Importance

Below are the references for the codified attribute numbers:

Attr58: total costs /total sales
Attr24: gross profit (in 3 years) / total assets
Attr46: (current assets - inventory) / short-term liabilities
Attr34: operating expenses / total liabilities
Attr27: profit on operating activities / financial expenses

The codified names were chosen to remain as the actual attributes themselves are all largely complex mathematical combinations that are equally as vague as just an attribute number.

For determining feature importance in a Random Forest we decided to use the MDI metric which stands for Mean Decrease in Impurity. The higher this value the better.

For our model it appears the most important attribute was Attr27 which is profit on operating activities / financial expenses. This attribute is a bit interesting as profit by itself already takes into account expenses as it is a measure of net gain. This attribute goes one step further by taking the net gain and producing a ratio of net gain to expenses. The main takeaway is the differences between gain of money and loss of money was useful to the model.

Attr34 and Attr46 both involve a metric with liabilities so that may be worth looking into further.

Attr 24 is a relationship between profit and assets. Profit was a component in the Attr27 so further evidence profit is important.

Attr58 is the ratio between costs and sales.

AUC Scores XG

```
XGB Cross Val AUC
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
auc=0.968272, std=0.004902
```

Figure 4: XGB 10 Fold Cross Validation AUC score and Standard Deviation

XGB achieved an auc score of 0.968. Additionally we can say with 95% confidence that the true auc score lies between 0.964 and 0.972. Since the confidence interval of random forest and auc don't overlap we can say statistically they have different values and that the XGB model is better. (RF: 0.944 and 0.956, XGB: 0.964 and 0.972)

Classification Report XG

XGB Classification Report

~~~~~~

	precision	recall	f1-score	support
0	0.98	1.00	0.99	4136
1	0.92	0.58	0.71	204
accuracy			0.98	4340
macro avg	0.95	0.79	0.85	4340
weighted avg	0.98	0.98	0.98	4340

Figure 5: XGB Classification Report

A value of 0 indicates the company did not go bankrupt while a value of 1 indicates a company did go bankrupt.

Overall the model performs well. Previously the random first struggled with the bankruptcy class but here we see the XGB model improved in this regard. It was able to achieve an f1 score of 0.71 compared to the random forest which got 0.48. This increase in performance was gained by improving recall but it came at a small cost of precision.

With both classification reports reviewed a small aside on metrics is worth discussing. The choice of auc as the optimizing metric was done because the paper we are comparing to used this metric. This appears to prefer optimizing precision over recall but in practice preferring recall is likely preferable here. For example if we were investing in a company and someone told us that the company will go bankrupt and it turned out down the road they never did that sucks but it's not a huge deal. On the other hand if they told us it wont go bankrupt and it ended up going bankrupt we may be frustrated with that result. The asymmetry in these failures is mirrored within the precision and recall metrics respectively.
~~~~~~

Feature Importance XG

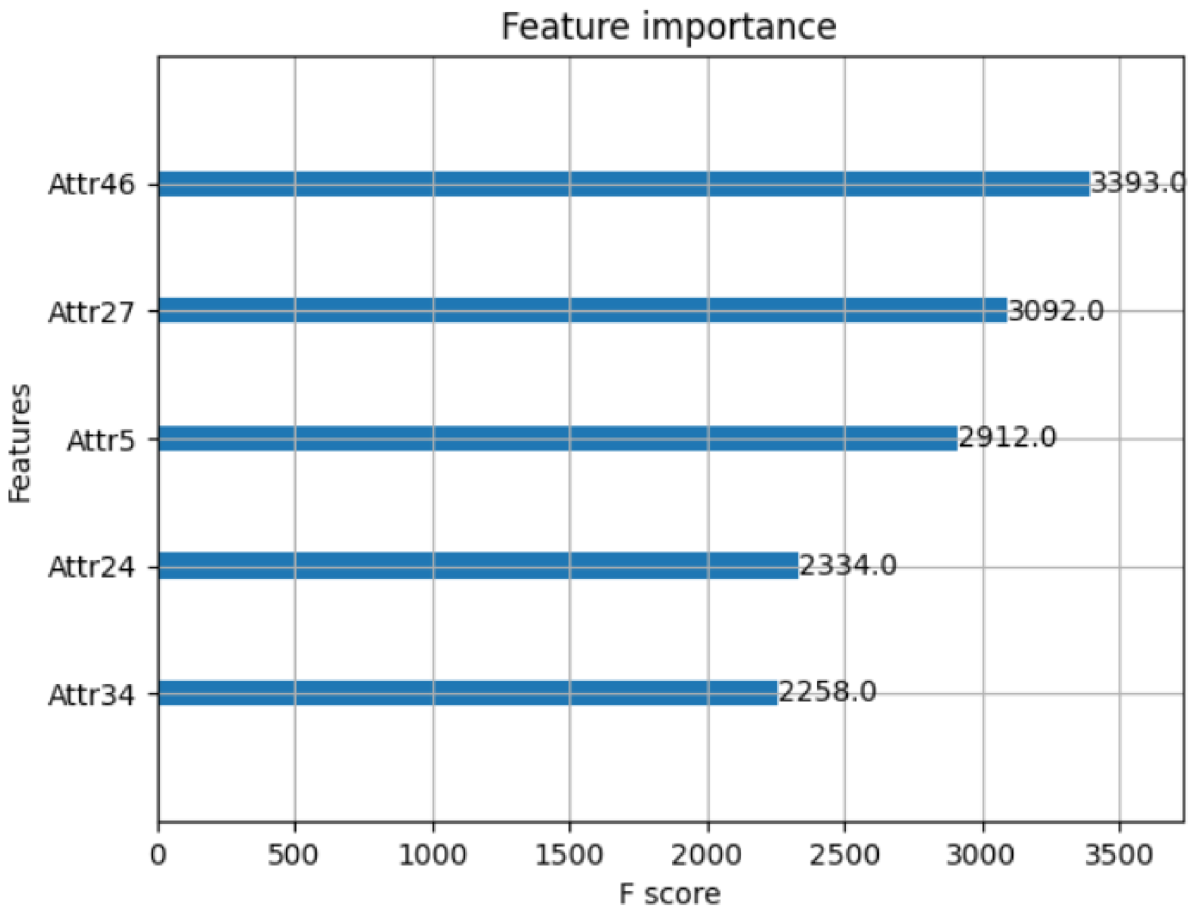

Figure 6: XGB Feature Importance

Below are the references for the codified attribute numbers:

Attr46: (current assets - inventory) / short-term liabilities
Attr27: profit on operating activities / financial expenses
Attr5: [(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365
Attr24: gross profit (in 3 years) / total assets
Attr34: operating expenses / total liabilities

First off it's important to observe that 4/5 of these features were also present in the top 5 features of the random forest albeit in a different order. This gives extra confidence that these features are important in bankruptcy prediction. A deeper exploration into the inherent behavior of these complex mathematical combinations is warranted.

For the XGB model the most important feature was Attr46 which is (current assets - inventory) / short-term liabilities. In the random forest model this was the third most important feature. This indicates that there is something important about this ratio but like stated earlier this is a complex relationship. A more thorough examination of this metric should be done to be able to derive why the model found this so important.

Since the other features were present in random forest the last attribute we will discuss is Attr5 which was not present in random forest and is the third most important feature for XGB. Attr5 is [(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365. This is an incredibly complex feature combination and is unlike any of the other features we've discussed. This is not easily interpreted but one might wonder if this was seen as important because it includes so many features within it that their cumulative impact is much larger than their individual parts.

ROC Comparison

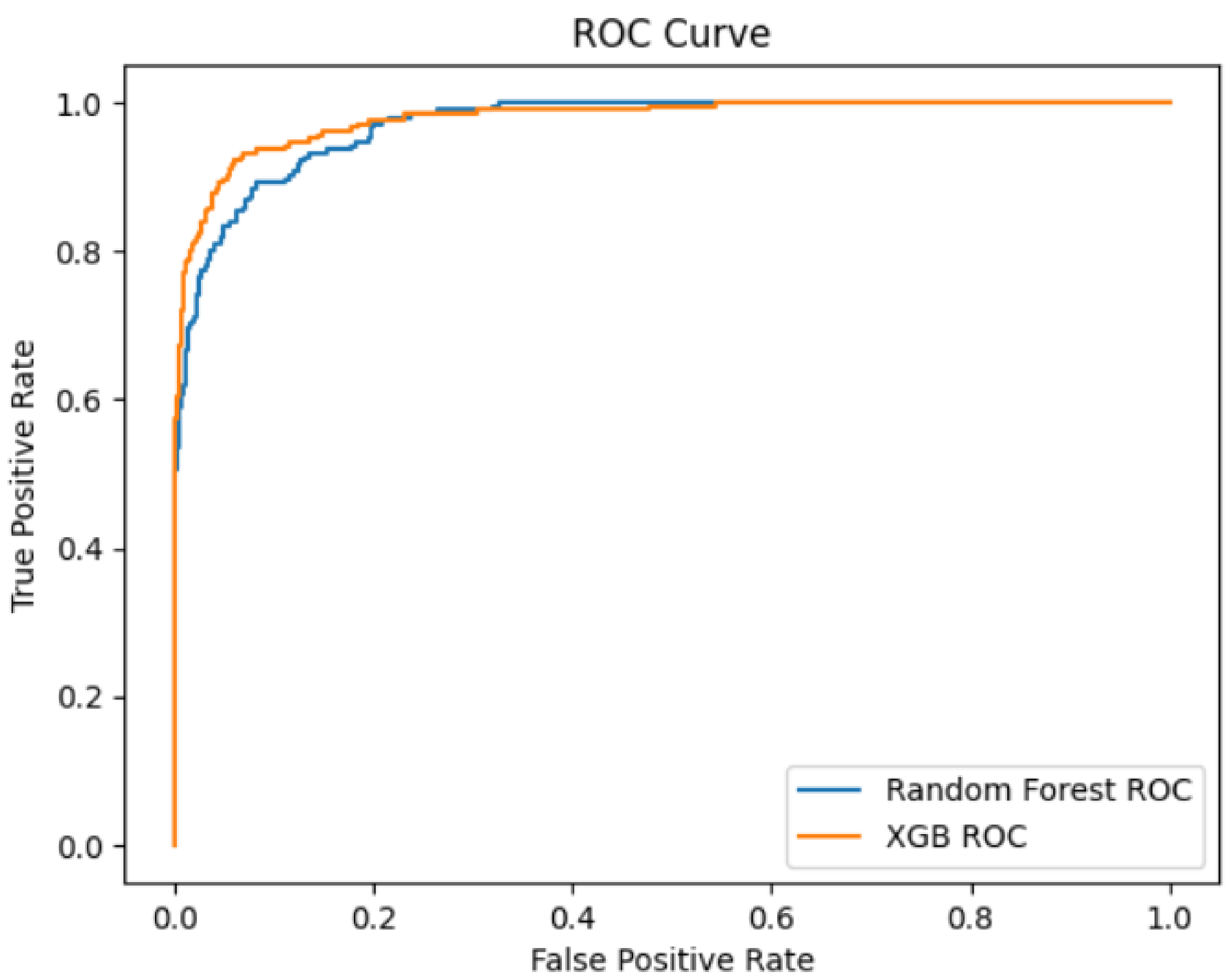

Figure 7: ROC Comparison

Lastly is an ROC comparison curve between the Random Forest and XGB model. It was observed earlier in the paper that XGB performs better so this is simply further justification.

Individually it also shows that both models can evenly trade off between true positive rate and false positive rate. This means there exists thresholds where we could prefer one metric doing better while slightly penalizing another. Previously we discussed optimizing recall over precision and while this plot shows TPR v FPR it can still somewhat help us as TPR is exactly recall. FPR is not exactly precision but we know that if we pick a threshold with higher TPR we would be directly improving recall.This comes at some unknown cost to precision but known cost to FPR.

Conclusion

This dataset was severely imbalanced and yet Random Forest and XGBoost handle this situation with ease. It wasn't mentioned in the report but the models with default settings also performed well but hyperparameter tuning helped a great deal with predicting bankruptcies (the rare case). Random Forest is a great model to start off a project as a base model as it is easy and quick to set up while XGBoost is for when performance is key and taking the time to maximize that is worth it.

Code

Appendix A: Custom XGBoost Random CV

```
def perform_optimal_search(
    clf_object,
    clf_hyperparams,
    static_params,
    optim_metric,
    x,
    y,
    n_iter=20,
    splits=2,
    report_freq=0.1
):

    kf = KFold(n_splits=splits)
    hyperparam_combinations = [
        dict(
            zip(clf_hyperparams, v)
        ) for v in product(*clf_hyperparams.values())
    ]
    shuffle(hyperparam_combinations)
    hyperparam_combinations_subset = sample(hyperparam_combinations, n_iter)
    report_freq = report_freq*len(hyperparam_combinations_subset)
    optimal_hyperparam = {}
    optimal_performance = {}
    first_iteration = True
    iter_number = 0
```

```python
for hyperparam in hyperparam_combinations_subset:
    performance = 0
    for id, (train_i, test_i) in enumerate(kf.split(x, y)):
        x_train = x.iloc[train_i]
        y_train = y.iloc[train_i]
        x_test = x.iloc[test_i]
        y_test = y.iloc[test_i]
        clf = clf_object(**static_params, **hyperparam)
        clf.fit(x_train, y_train, eval_set=[(x_test, y_test)], verbose=False)
        y_pred = clf.predict_proba(x_test)[:, 1]
        performance += optim_metric(y_test, y_pred)
    performance = performance/splits
    if first_iteration:
        first_iteration = False
        optimal_hyperparam = hyperparam
        optimal_performance = performance
    else:
        if performance > optimal_performance:
            optimal_performance = performance
            optimal_hyperparam = hyperparam
    if (iter_number+1) % report_freq == 0:
        percent_done = (iter_number+1)/len(hyperparam_combinations_subset)*100
        print(f"Precent Done: {percent_done:4f}%")
    iter_number += 1

return optimal_performance, optimal_hyperparam
```

Appendix B: Script

```python
import sys
import os
from scipy.io import arff
import pandas as pd
from sklearn.impute import SimpleImputer
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
import sklearn.metrics as mt
import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from xgboost import plot_importance
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import roc_auc_score


class UnexpectedFileStructureError(Exception):
    def __init__(self):
        msg = "This script expects a file path to a directory that contains "
        msg += "CS4 Bankruptcy Dataset. The Bankruptcy dataset is a "
        msg += "group of five arff files with the names, 1year, 2year, "
        msg += "3year, 4year, 5year. Did you specify a folder path that "
        msg += "contains these files?"
        super().__init__(msg)


```

```python
class MissingInputError(Exception):
    def __init__(self):
        msg = "This script expects exactly one input, a file directory that "
        msg += "contains the bankruptcy dataset. You input nothing. "
        msg += "Did you forget to add an argument with the script?"
        super().__init__(msg)


class ExcessiveInputError(Exception):
    def __init__(self):
        msg = "This script expects exactly one input, a file directory that "
        msg += "contains the bankruptcy dataset. You called this script "
        msg += "with more than one argument. Only use one and try again. "
        super().__init__(msg)


def check_correct_file_structure(inputs):
    data_direc = inputs[1]
    expected_files = ["1year.arff",
                      "2year.arff",
                      "3year.arff",
                      "4year.arff",
                      "5year.arff"]
    for root, dirs, files in os.walk(data_direc):
        for f in files:
            if f not in expected_files:
                print(f)
                raise UnexpectedFileStructureError

```

```python
def check_valid_inputs(inputs):
    if len(inputs) < 2:
        raise MissingInputError
    elif len(inputs) > 2:
        raise ExcessiveInputError


def load_data(inputs):
    data_direc = inputs[1]
    raw_year_data = []
    for root, dirs, files in os.walk(data_direc):
        for f in files:
            file_path = os.path.join(root, f)
            arff_data = arff.loadarff(file_path)
            raw_year_data.append(pd.DataFrame(arff_data[0]))
    return raw_year_data


def log_section(title="No Title", content="No Content"):
    log_file.write(title + "\n")
    log_file.write("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")
    log_file.write(content + "\n")
    log_file.write("\n")

```

```python
def preprocess_data(raw_df):
    # Seperate features by percentage of data NA
    na_variations = [np.nan]
    below_50_precent_na_features = []  # Mean Impute
    above_50_perecent_na_features = []  # Remove

    for feature in list(raw_df):
        percent_na = raw_df[feature].isin(na_variations).sum()
        percent_na = percent_na / (raw_df.shape[0]) * 100
        percent_na = round(percent_na, 3)
        if percent_na <= 50:
            below_50_precent_na_features.append(feature)
        if percent_na > 50:
            above_50_perecent_na_features.append(feature)

    # Handle NAs
    clean_df = raw_df.copy()
    # Mean Inpute if less than 50% missing
    mean_imp = SimpleImputer(missing_values=np.nan, strategy='mean')
    clean_df[below_50_precent_na_features] = mean_imp.fit_transform(
        clean_df[below_50_precent_na_features])

    # Remove if more than 50% missing
    for feature in above_50_perecent_na_features:
        clean_df = clean_df.drop([feature], axis=1)

    # Changing Target Feature to be int type
    target_feature = ["class"]
    cts_features = []
    for feature in list(clean_df):
        if feature not in target_feature:
            cts_features.append(feature)
    clean_df[target_feature] = clean_df[target_feature].astype(int)

    return clean_df
```

```python
if __name__ == '__main__':
    inputs = sys.argv
    check_valid_inputs(inputs)
    check_correct_file_structure(inputs)
    print("Full Execution on my machine takes around 20 Minutes")
    # Log Start
    log_file = open("log.txt", "w")
    log_file.write("Case Study 4 Report\n\n")

    # Use input to load data
    raw_years = load_data(inputs)

    # Group all years into one large dataset and shuffle
    raw_df = pd.concat(raw_years, axis=0, ignore_index=True)

    # Data Preprocessing
    clean_df = preprocess_data(raw_df)

    # Target Feature Mappings
    bankrupt_to_id_map = {"Not Bankrupt": 0, "Bankrupt": 1}
    id_to_bankrupt_map = {0: "Not Bankrupt", 1: "Bankrupt"}

    # Shuffle
    clean_df = clean_df.sample(frac=1)
```

```python
    # Seperate Features and Target
    target_feature = ["class"]
    y = clean_df[target_feature[0]]
    x = clean_df.drop(target_feature, axis=1)

    ninety_precent_of_data = int(x.shape[0]*0.9)
    ten_precent_of_data = int(x.shape[0]*0.1)
    train_x = x.head(ninety_precent_of_data)
    train_y = y.head(ninety_precent_of_data)
    val_x = x.tail(ten_precent_of_data)
    val_y = y.tail(ten_precent_of_data)

    # Random Forest Model
    rf_params = {'min_weight_fraction_leaf': 3.359818286283781e-06,
                 'min_samples_split': 10,
                 'min_samples_leaf': 6,
                 'min_impurity_decrease': 2.069138081114479e-05,
                 'max_depth': 429,
                 'criterion': 'entropy',
                 'n_jobs': 4,
                 'n_estimators': 1000}

    # Cross Val Score
    rf = RandomForestClassifier(**rf_params)
    splits = KFold(n_splits=10, shuffle=True)
    cross_score_rf = cross_val_score(rf, x, y, cv=splits, scoring='roc_auc')
    auc_rf = cross_score_rf.mean()
    auc_rf_std = cross_score_rf.std()
    rf_cross_val_rep = f"auc={auc_rf:4f}, std={auc_rf_std:4f}"
    log_section(title="RF Cross Val AUC",
                content=rf_cross_val_rep)
```

```
    # XGB Model
    xg_params = {
        "objective": "binary:logistic",
        "nthread": 4,
        "eval_metric": "auc",
        "early_stopping_rounds": 5,
        "n_estimators": 200,
        'eta': 0.6309573444801936,
        'max_depth': 12,
        'min_child_weight': 0.00630957344480193,
        'subsample': 1.0,
        'colsample_bytree': 1.0,
        'colsample_bylevel': 1.0,
        'colsample_bynode': 1.0,
        'lambda': 10}

    # Cross Val Score
    cross_score_xgb = []
    split_count = 10
    kf = KFold(n_splits=split_count)
    for id, (train_i, test_i) in enumerate(kf.split(x, y)):
        x_train = x.iloc[train_i]
        y_train = y.iloc[train_i]
        x_test = x.iloc[test_i]
        y_test = y.iloc[test_i]
        xg_model = XGBClassifier(**xg_params)
        xg_model.fit(x_train,
                     y_train,
                     eval_set=[(x_test, y_test)],
                     verbose=False)
        y_pred = xg_model.predict_proba(x_test)[:, 1]
        cross_score_xgb.append(roc_auc_score(y_test, y_pred))
```

```python
    cross_score_xgb = np.asarray(cross_score_xgb)
    auc_xgb = cross_score_xgb.mean()
    auc_xgb_std = cross_score_xgb.std()
    xgb_cross_val_rep = f"auc={auc_xgb:4f}, std={auc_xgb_std:4f}"
    log_section(title="XGB Cross Val AUC",
                content=xgb_cross_val_rep)

    # Final Model For Analysis
    xg_params_no_early = {
        "objective": "binary:logistic",
        "nthread": 4,
        "n_estimators": 200,
        'eta': 0.6309573444801936,
        'max_depth': 12,
        'min_child_weight': 0.0063095734448019З,
        'subsample': 1.0,
        'colsample_bytree': 1.0,
        'colsample_bylevel': 1.0,
        'colsample_bynode': 1.0,
        'lambda': 10}
    xg_model = XGBClassifier(**xg_params_no_early)
    xg_model.fit(train_x, train_y, verbose=False)

    # Random Forest Analysis
    # Classification Report
    y_pred_final = rf.predict(val_x)
    y_true = val_y
    classif_rep = classification_report(
        y_true,
        y_pred_final
    )
    log_section(title="Random Forest Classification Report",
                content=classif_rep)
```

```python
    # AUC and ROC Curve All Data
    y_pred_prob = rf.predict_proba(val_x)
    pos_class = 1
    y_label = id_to_bankrupt_map[pos_class]
    preds = y_pred_prob[:, pos_class]
    fpr_tree, tpr_tree, thresholds = mt.roc_curve(val_y,
                                                  preds,
                                                  pos_label=pos_class)
    auc_report = f"{y_label} AUC: {mt.auc(fpr_tree, tpr_tree):.4f}\n"
    log_section(title="Random Forest AUC Report",
                content=auc_report)

    # Feature Importance
    importances_rf = pd.DataFrame(
        rf.feature_importances_,
        index=x.columns,
        columns=["importance"]).sort_values("importance", ascending=False)
    fig, ax = plt.subplots()
    importances_rf.head(5).plot.barh(ax=ax)
    ax.set_title("Feature importances using MDI")
    ax.set_ylabel("Mean decrease in impurity")
    plt.savefig("rf_feature_importance.png", bbox_inches='tight')
    plt.clf()

    # XGB Analysis
    # Classification Report
    y_pred_final = xg_model.predict(val_x)
    y_true = val_y
    classif_rep = classification_report(
        y_true,
        y_pred_final
    )
    log_section(title="XGB Classification Report",
                content=classif_rep)
```

```python
    # AUC and ROC Curve All Data
    y_pred_prob = xg_model.predict_proba(val_x)
    pos_class = 1
    y_label = id_to_bankrupt_map[pos_class]
    preds = y_pred_prob[:, pos_class]
    fpr_xgb, tpr_xgb, thresholds = mt.roc_curve(val_y,
                                               preds,
                                               pos_label=pos_class)
    auc_report = f"{y_label} AUC: {mt.auc(fpr_xgb, tpr_xgb):.4f}\n"
    log_section(title="XGB AUC Report",
                content=auc_report)

    # Feature Importance
    plot_importance(xg_model, max_num_features=5)
    plt.savefig("xgb_feature_importance.png", bbox_inches='tight')
    plt.clf()

    # ROC Comparison Plot
    plt.plot(fpr_tree, tpr_tree, label="Random Forest ROC")
    plt.plot(fpr_xgb, tpr_xgb, label="XGB ROC")
    plt.title("ROC Curve")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.savefig("roc_plot.png", bbox_inches='tight')
    plt.clf()

    log_file.write("\n")
    log_file.close()
```