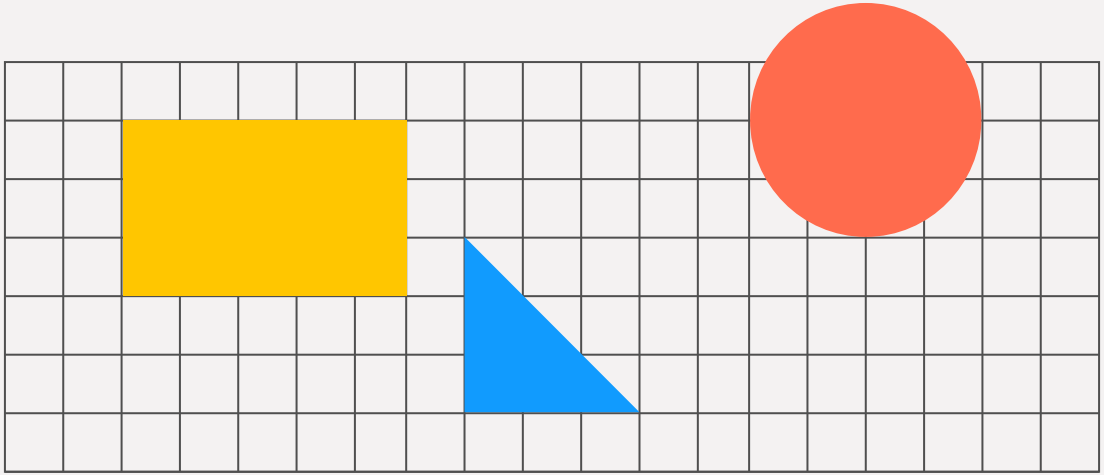
ML 1
Final Project

Predictive Maintenance Classification Models



Problem Statement:

Unplanned machine failures cause significant downtime and lost revenue for the businesses operating them.

Goal:

Build a classifier model that can predict machine failures to a high enough level of accuracy to provide business utility.

Dataset Overview:

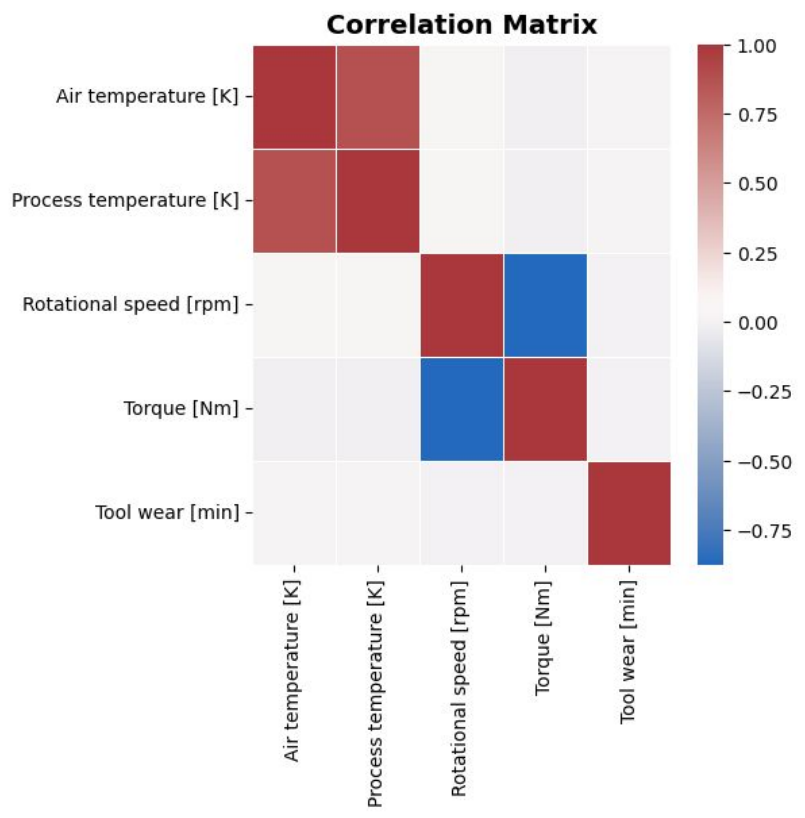
A predictive maintenance dataset provided by UC Irvine via kaggle was selected for this problem.

The dataset has 10,000 entries and 10 columns.

7 of the columns are numeric, 3 are not, I have excluded the non-numeric columns from further analysis.

The outcome variable “Target” is a binary value {0,1} where 0 is a healthy machine and 1 is a machine failure. The data is highly imbalanced with 3.4% of the entries being machine failures.

Correlation Matrix



Strong correlations are observed between a few variables, mainly:

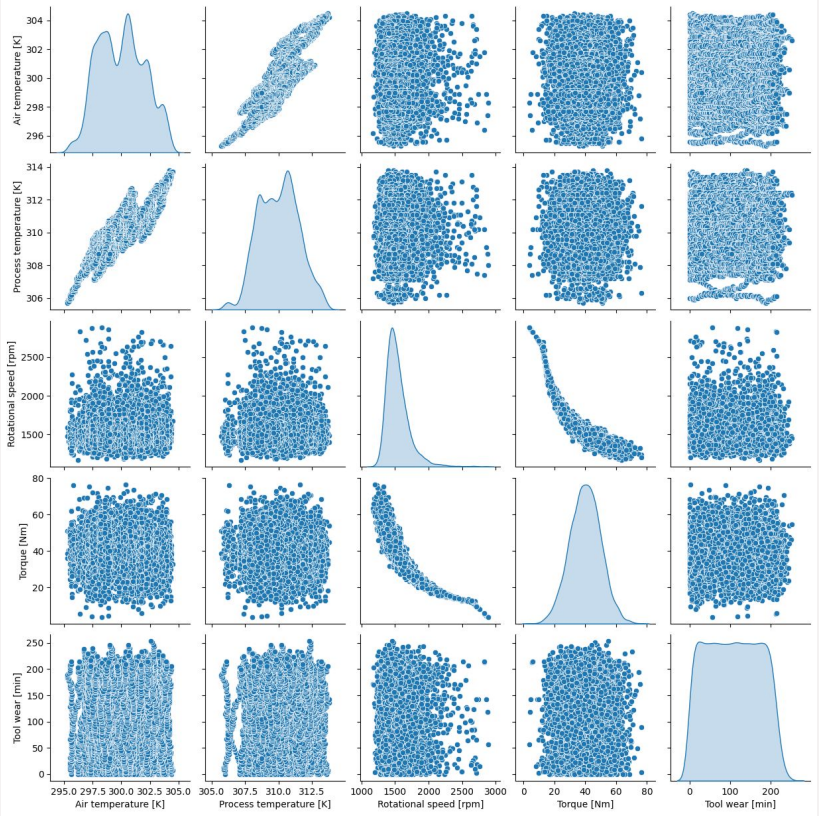
Air Temperature - Process Temperature

Torque - Rotational Speed

Both of these relationships can be explained via the physics equations governing their behavior.

In order to reduce collinearity, I have excluded Air Temperature & Rotational Speed

Pair Plot



The pair plots further reinforce the relationships seen in the the correlation matrix.

The main information of note within the pair plot is the distribution of each feature. Tool wear has a larger, more uniform distribution, where as the other features have more clustered values.

Testing Train Split:

Using the information from the correlation and pair plots. I decided that my features will be: Process temperature [K], Torque [Nm], and Tool wear [min].

After selecting these parameters, I split the data into testing and training sets reserving 20% of the data for test & verification work.

```
# Setting up my test train split, will be using this split to evaluate all following models
x = maintenance_df[["Process temperature [K]", "Torque [Nm]", "Tool wear [min]"]]
y = maintenance_df["Target"]
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2)
```

Models Tested:

I chose to approach this problem by experimenting with three different decision tree classifiers:

1. Random Forest Classifier
2. Adaboost Classifier
3. Gradient Boost Classifier

While we did not learn about gradient boost in class, I wanted to try a new model in addition to two known models.

I chose these models because they can perform well in nonlinear applications and I was not sure at first inspection what the nature of the relationship between the features and the outcome was.

Grading Criteria

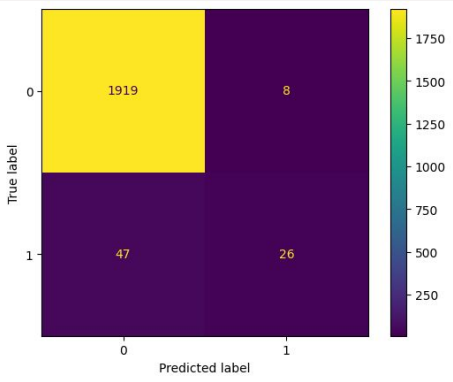
Due to the highly imbalanced nature of the dataset, and the potential large costs to a business of having a machine failure fail to be caught, Recall will be to most important metrics on which the models are graded.

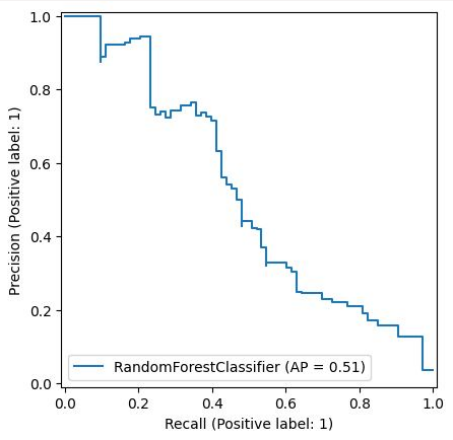
A high number of false positives may be tolerable to the end user if it yields better performance catching true positives.

I will be using confusion matrices and Precision - Recall curves to grade every iteration of the models.

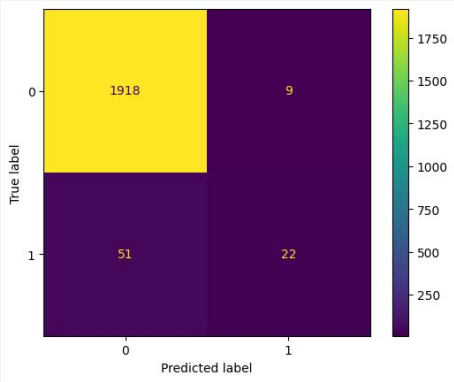
Initial Results

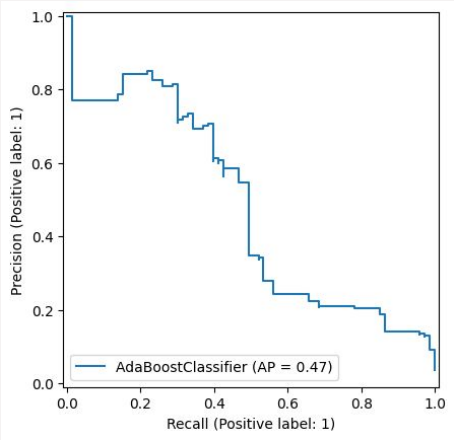
Random Forest



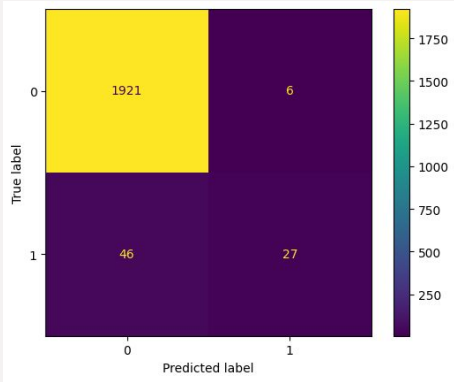


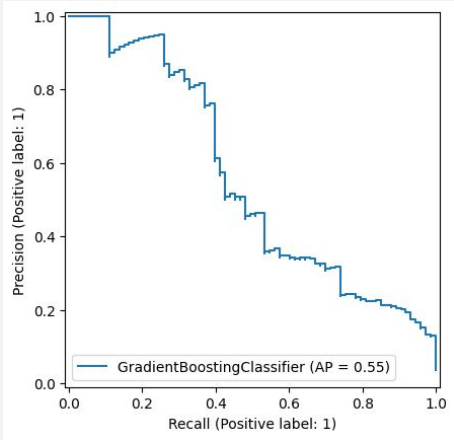
Adaboost





Gradient Boost

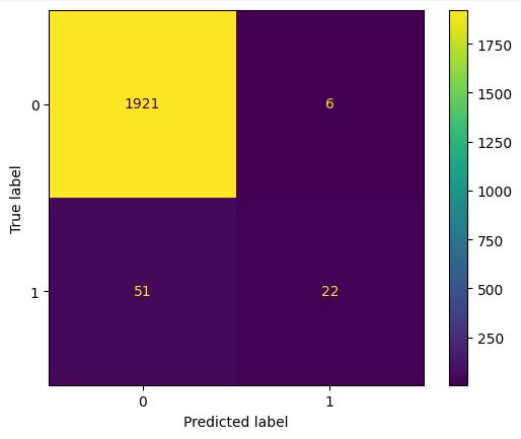


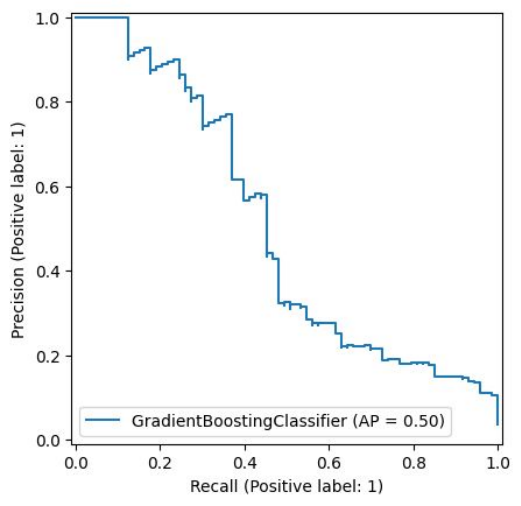


Grid Search

I employed a grid search with a 5 fold cross-validation in order to try to tune the hyperparameters and gain better model performance for the Gradient Boost model.

```python
# This is the grid search parameters I will be using to try to refine my gradient boosted model
param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 4],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
}

# Setting up a 5 fold cv grid search
grid_search = GridSearchCV(estimator=gradient_model, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Extracting the best model from the grid search for use
best_gradient_model = grid_search.best_estimator_

ConfusionMatrixDisplay.from_estimator(best_gradient_model, X_test, y_test)
PrecisionRecallDisplay.from_estimator(best_gradient_model, X_test, y_test)
plt.show()
```





SMOTE Approach:

After going through the prior steps, it became clear that the imbalance in the training data would need to be addressed in order to yield a better model.

To accomplish this I employed SMOTE. SMOTE is an approach the creates synthetic entries of the minority data (in this case machine failures) to create a more balanced training dataset.

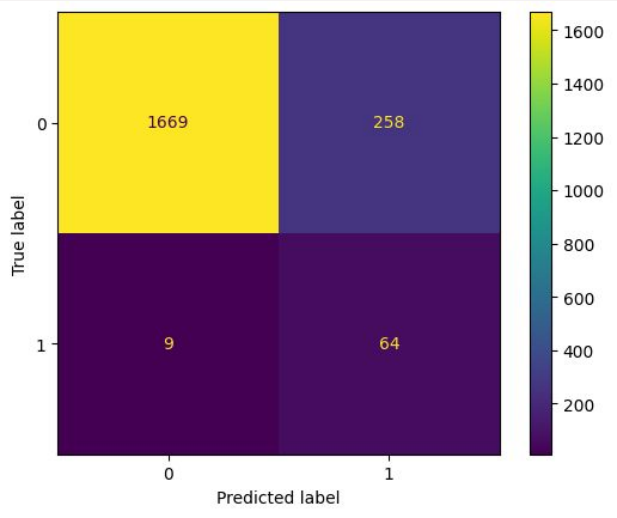
```
# Setting up smote to sample my data and create a new balanaced set

smote = SMOTE(sampling_strategy='auto', random_state=2)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)


✓  0.0s
```

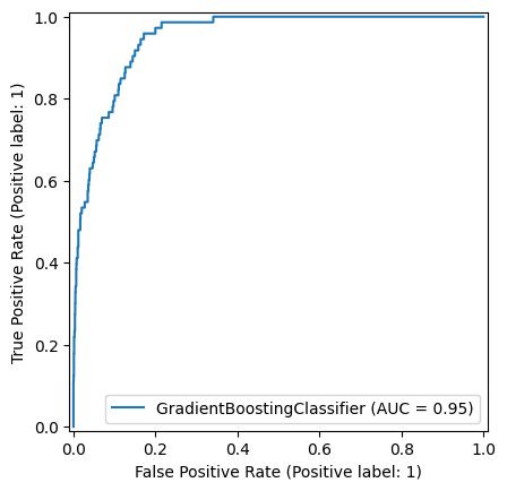
SMOTE Results:

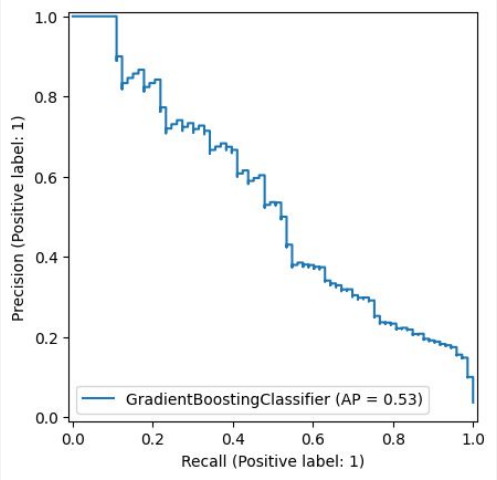
The SMOTE training data was used to retrain all 3 models as can be seen in the notebook. The gradient boosted model performed best and will be discussed here.

The model now has far better recall performance, at the expense of an increase in false positives. This may be an acceptable trade-off depending on the business case being studied.

While less of a metric of consideration due to the nature of the data, the model performs well when looking at the TPR FPR curve, with an AUC of 0.95







Conclusions:

The Gradient Boosted model, using SMOTE data produces the best results in testing. That model can be expected to have many false positives if used in production but the false positive rate may be acceptable to the business.

What Didn't Work:

The grid search actually yielded a worse model than the initial model. This is likely due to too small of a search range. With more computational power, I would be able to run a larger amount of options through the grid search process and potentially yield a better model.

Directions for Improvement

1. Better hyperparameter tuning via giving more compute power to a larger grid search process and picking better values for the ranges.
2. Exploring SVM and Logistic Regression models. These models, with efficient tuning, may produce better results than the models tried here.
3. Gathering more data. Further test and data acquisition campaigns could yield a larger population that allows for better model performance when trained.

Sources:

AI4I 2020 Predictive Maintenance Dataset [Dataset]. (2020). UCI Machine Learning Repository. https://doi.org/10.24432/C5HS5C.

Github:

https://github.com/pekeren14/Supervised_Learning_Final