AI4I Binary Classification Prediction

Ying Yiwen, 12210159

Content

1

Introduction

AI4I 2020 Predictive Maintenance Dataset

- real predictive maintenance encountered in industry
- feature: Type,Air temperature [K],Process temperature [K],Rotational speed [rpm],Torque [Nm],Tool wear [min]
- label: Machine failure,~~TWF,HDF,PWF,OSF,RNF~~
- 9661 0-class and 339 1-class
- linear indivisible
- features are not independent of each other

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

Code Organization

- **load.py** - load dataset, deal with rows and columns
- **preprocess.py** - upsampling, downsampling
- **linear_regression.py** - class LinearRegression
- **perceptron.py** - class Perceptron
- **logistic _regression.py** - class LogisticRegression
- **multi_layer_perceptron.py** - class MultiLayerPerceptron
- **evaluation.py** - evaluate the model and visualize
- **main.py** - entrance to the code

Improvment Idea

- data preprocessing
  - upsampling and downsampling to balance the class
- feature engineering
  - combination of different feature, with prior knowledge with such practical scenarios
- model optimizer
  - choose suitable activation function, loss function
  - momemtum tuning
  - adaptive learning rate
  - dropout regularization
- hyperparameters

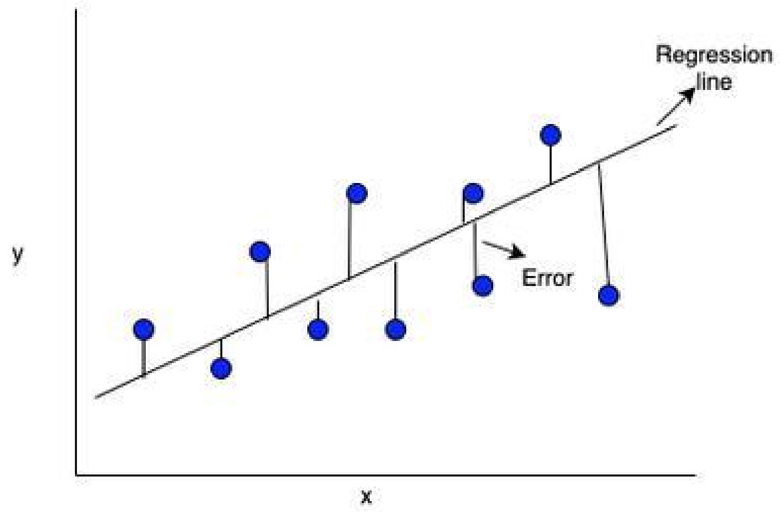
“2”

Principle - Basic Model

Linear Regression

- $y = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n + \varepsilon$
- $\omega \leftarrow \omega + \lambda \frac{1}{m} X_{B_i}^T (t_{B_i} - X_{B_i}\omega)$

```
def train(self, X_train, y_train, X_test, y_test):
    # initialize weights
    self.W = np.random.rand(X_train.shape[1] + 1)
    N = y_train.size
    for _ in range(self.n_iter):
        # shuffle the data
        indices = np.random.permutation(N)
        for start in range(0, N, self.batch_size):
            end = min(start + self.batch_size, N)  # not out of range
            # get batch data
            batch_indices = indices[start:end]
            X_batch = X_train[batch_indices]
            y_batch = y_train[batch_indices]
            # predict and compute loss
            y_pred = self.predict(X_batch)
            train_loss = self.calculate_loss(y_batch, y_pred)
            self.train_loss.append(train_loss)
            y_pred_test = self.predict(X_test)
            test_loss = self.calculate_loss(y_test, y_pred_test)
            self.test_loss.append(test_loss)
            # compute gradient
            grad = self.gradient(X_batch, y_batch, y_pred)
            self.W -= self.lr * grad
```



Perceptron

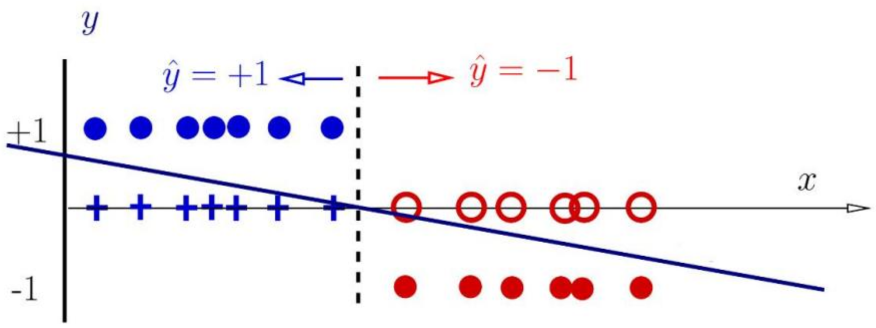
- $y = sign(\omega \cdot x + b)$
- $\omega \leftarrow \omega - \eta(y_{i-true} - y_{i-pred})x_i$

解释代码 | 注释代码 | 生成单测 | ×

```python
def _loss_batch(self, y, y_pred):
    # Weighted hinge loss for a batch with L2 regularization
    weights = np.where(y == 1, self.positive_weight, 1 - self.positive_weight)
    hinge_loss = np.maximum(0, -y * y_pred) * weights
    reg_loss = self.alpha * np.sum(self.W[1:] ** 2)  # Exclude bias term from regularization
    return hinge_loss.mean() + reg_loss
```

解释代码 | 注释代码 | 生成单测 | ×

```python
def _gradient_batch(self, X, y, y_pred):
    # Gradient of weighted hinge loss for a batch with L2 regularization
    weights = np.where(y == 1, self.positive_weight, 1 - self.positive_weight)
    misclassified = y_pred * y < 0
    gradient = -(X[misclassified].T @ (weights[misclassified] * y[misclassified])) / X.shape[0]
    gradient[1:] += 2 * self.alpha * self.W[1:]  # Apply L2 regularization (exclude bias)
    return gradient
```



Logistic Regression

- $P(y=1|X) = \frac{1}{1+e^{-(w \cdot x+b)}}$

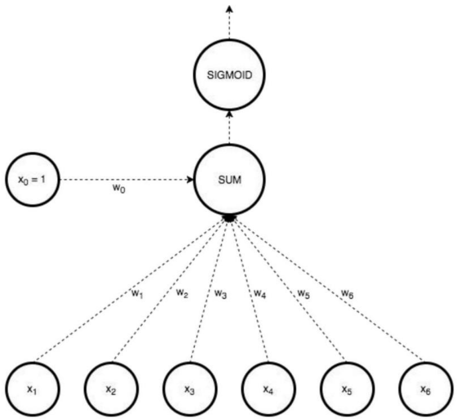
- $L(w) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_{i-true}\log y_{i-pred} + (1-y_{i-true})\log\left(1-y_{i-pred}\right)\right]$

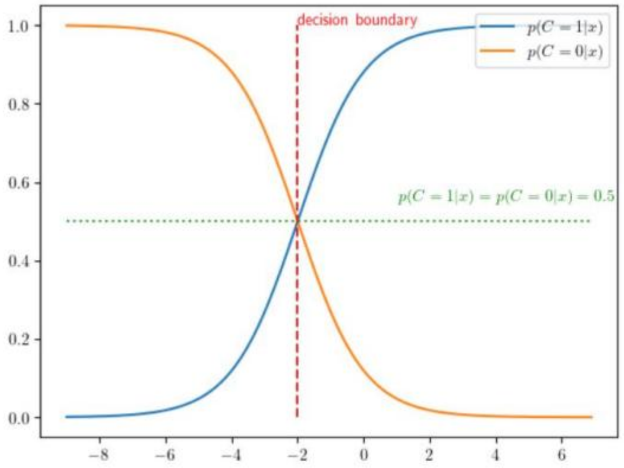
```python
@staticmethod
解释代码 | 注释代码 | 生成单测 | ×
def _softplus(x):
    return np.log(1 + np.exp(x)) / (1 + np.log(1 + np.exp(x)))

解释代码 | 注释代码 | 生成单测 | ×
def predict_probability(self, X):
    return self._softplus(X @ self.W)

@staticmethod
解释代码 | 注释代码 | 生成单测 | ×
def _loss(y, y_pred, epsilon=1e-5):
    # Weighted cross entropy loss
    weights = np.where(y == 1, 0.5, 0.5)
    loss = -weights * (y * np.log(y_pred + epsilon) + (1 - y) * np.log(1 - y_pred + epsilon))
    return np.mean(loss)

解释代码 | 注释代码 | 生成单测 | ×
def _gradient(self, X, y, y_pred):
    # Weighted gradient for cross entropy loss
    weights = np.where(y == 1, 0.6, 0.5)
    reg_term = self.alpha * self.W  # Regularization term
    weighted_diff = weights * (y_pred - y)
    return (weighted_diff @ X) / y.size + reg_term
```





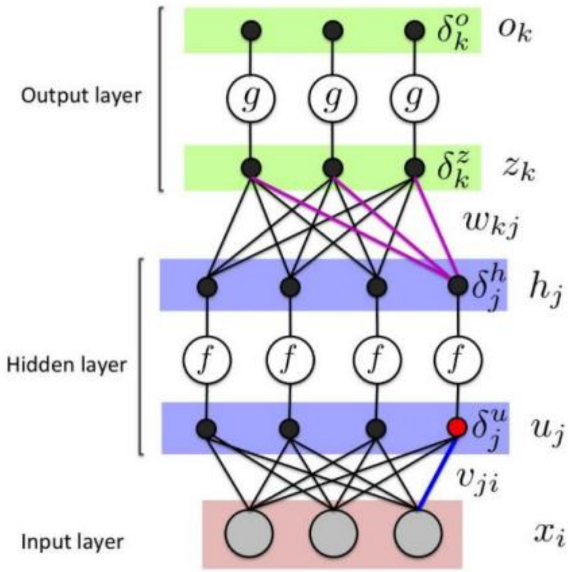
Multi-Layer Perceptron

- $h^{(l)} = \sigma(W^{(l)} h^{(l-1)} + b^{(l)})$
- $W \leftarrow W - \eta \frac{\partial E}{\partial W}$

```
def backward(self, inputs, targets, epoch):
    m = inputs.shape[0]
    predictions = self.activations[-1]
    delta = predictions - targets

    # Update output layer
    grad_w = np.dot(self.activations[-2].T, delta) / m + 2 * self.l2_lambda * self.weights[-1]
    grad_b = np.sum(delta, axis=0, keepdims=True) / m
    self.update_params(-1, grad_w, grad_b, epoch)

    # Backpropagate through hidden layers
    for i in range(self.num_layers - 2, 0, -1):
        delta = np.dot(delta, self.weights[i].T) * self.activation_derivative(self.z_values[i - 1])
        grad_w = np.dot(self.activations[i - 1].T, delta) / m + 2 * self.l2_lambda * self.weights[i - 1]
        grad_b = np.sum(delta, axis=0, keepdims=True) / m
        self.update_params(i - 1, grad_w, grad_b, epoch)
```



“2”

Principle - Optimization

Dataset Condition —— Bad!

- very unbalanced, 9661 0-class, 339 1-class
- training leans to 0-class condition
- performance of test size lose effectiveness (only tells 0-class)

- upsampling, ADASYN
- undersampling, cluster-based deletion

ADASYN Upsampling

Algorithm 2 ADASYN Algorithm

Input: Dataset (X, y), minority class label, k neighbors, balance ratio β
Output: Resampled dataset $(X_{\text{resampled}}, y_{\text{resampled}})$

```
Split X into X_minority and X_majority
Compute number of samples to generate: G = β × (#majority − #minority)
Calculate difficulty for each minority sample using k-nearest neighbors
Normalize difficulty to get sampling weights
for each minority sample x_i do
    Generate g_i synthetic samples:
    Randomly select a neighbor and create new samples along the line
end for
Append synthetic samples to the original dataset
return (X_resampled, y_resampled)
```

Cluster-Based Undersampling

Algorithm 3 Cluster-Based Undersampling Algorithm

Input: Dataset (X, y), undersampling ratio ratio

Output: Resampled dataset $(X_{\text{resampled}}, y_{\text{resampled}})$

1: Identify majority and minority classes based on y

2: Split X into X_{majority} and X_{minority}

3: Compute target majority class size: $n_{\text{majority\_target}} = \frac{n_{\text{minority}}}{(1-\text{ratio})} - n_{\text{minority}}$

4: Apply K-Means clustering on X_{majority} with $n_{\text{majority\_target}}$ clusters

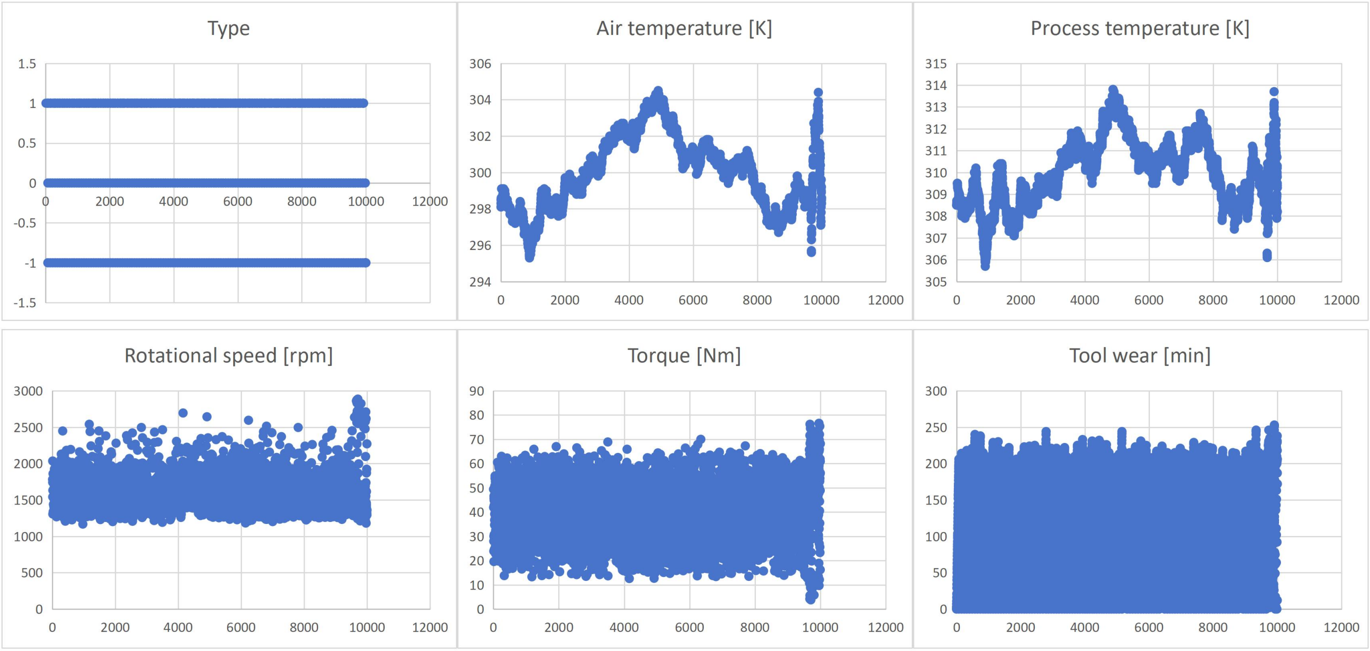
5: Select one representative sample (nearest to cluster center) from each cluster

6: Combine X_{minority} with the representative samples

7: **return** $(X_{\text{resampled}}, y_{\text{resampled}})$

Dataset Condition —— Bad!

- Feature can’t tell label clearly by itself, especially linear combination
- Following graph: first 9661 0-class, last 339 1-class
- No clear distinction!



Data Meaning?

- The machine failure consists of five independent failure modes:

- tool wear failure (TWF): the tool will be replaced of fail at a randomly selected tool wear time between 200 â€“ 240 mins (120 times in our dataset). At this point in time, the tool is replaced 69 times, and fails 51 times (randomly assigned).
- heat dissipation failure (HDF): heat dissipation causes a process failure, if the **difference between air- and process temperature** is below 8.6 K and the toolâ€™s rotational speed is below 1380 rpm. This is the case for 115 data points.
- power failure (PWF): the **product of torque and rotational speed** (in rad/s) equals the power required for the process. If this power is below 3500 W or above 9000 W, the process fails, which is the case 95 times in our dataset.
- overstrain failure (OSF): if the product of **tool wear and torque exceeds 11,000 minNm for the L product variant** (12,000 M, 13,000 H), the process fails due to overstrain. This is true for 98 datapoints.
- random failures (RNF): each process has a chance of 0,1 % to fail regardless of its process parameters. This is the case for only 5 datapoints, less than could be expected for 10,000 datapoints in our dataset.

Feature Engineering!

```
# Step 3: Replace 'Type' column values
pd.set_option('future.no_silent_downcasting', True)
data_cleaned['Type'] = data_cleaned['Type'].replace({'L': 11, 'M': 12, 'H': 13}).infer_objects(copy=False).astype('float64')

# Step 4: Add new features
data_cleaned['AirTemp_ProcessTemp'] = data_cleaned['Air temperature [K]'] - data_cleaned['Process temperature [K]']
data_cleaned['RotSpeed_Torque'] = data_cleaned['Rotational speed [rpm]'] * data_cleaned['Torque [Nm]']
data_cleaned['Torque_ToolWear_TypeL'] = data_cleaned['Torque [Nm]'] * data_cleaned['Tool wear [min]'] * data_cleaned['Type']
```

Activation Function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\mathrm{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

$$\mathrm{Softplus}(x) = \log(1 + e^{x})$$

$$\boxed{\mathrm{ReLU}(x) = \max(0, x)}$$

$$\mathrm{GELU}(x) = x \cdot \Phi(x)$$

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

TRY!

Loss Function

- Unbalanced Dataset → Weighted Loss Function

$$\text{Weighted MSE} = \frac{1}{n}\sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2$$

$$\text{Weighted Binary Cross-Entropy Loss} = -\frac{1}{n}\sum_{i=1}^{n} w_i \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right]$$

Dynamic Hyperparameter

- learning rate
- too large, diverge
- too small, converge slow / local minimum
- single lr can't balance efficiency and stability

```
self.lr = 0.9999 * self.lr
```

$$0.9999^{10000} \approx 0.3679$$

Dynamic Hyperparameter

- update method, considering history, more smooth
- momemtum tuning

$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta_t)$$

$$\theta_{t+1} = \theta_t - v_t$$

where v_t is current momemtum, γ is momemtum coefficient, η is learning rate, $\nabla J(\theta_t)$ is current gradient.

```
self.velocity = self.momentum * self.velocity + self.lr * grad
self.W -= self.velocity
```

Adam Optimizer

- first order momemtum and second order momemtum

$$\text{Gradient } g_t = \nabla J(\theta_t)$$

$$\text{First Order Momemtum } m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$\text{Second Order Momemtum } v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\text{Bias-Corrected Values } \hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\text{Update Method: } \theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- g_t is the gradient of the loss function $J(\theta)$ at time step t.
- m_t and v_t are the first-order and second-order moment estimates, respectively.
- β_1 and β_2 are the exponential decay rates for the moment estimates, typically set to 0.9 and 0.999.
- η is the learning rate.
- ϵ is a small constant (e.g., 10^{-8}) to avoid division by zero.

Dropout

- Prevent overfitting in MLP
- randomly drop neurons
- higher generalization

```
def dropout(self, x):
    if self.training:
        mask = np.random.rand(*x.shape) > self.dropout_rate
        return x * mask / (1 - self.dropout_rate)
    return x
```

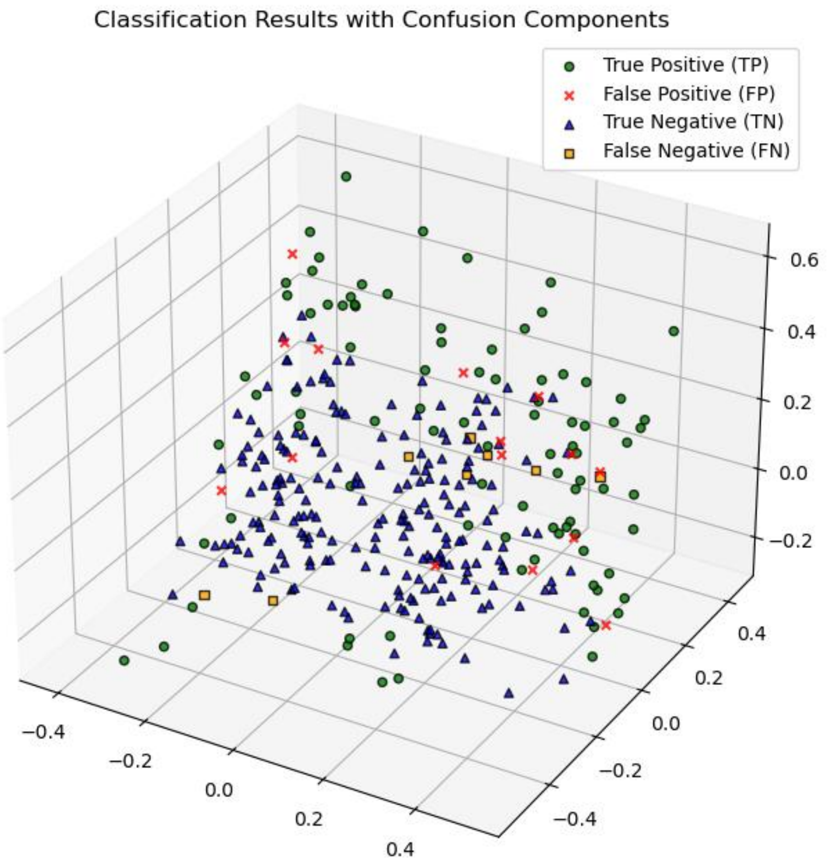
Evaluation

- True Positive (TP): the number of positive classes that the model correctly predicts as positive.
- False Positive (FP): the number of negative classes that the model incorrectly predicts as positive.
- True Negative (TN): the number of negative classes that the model correctly predicts as negative.
- $\boxed{\text{False Negative (FN)}}$: the number of negative classes that the model incorrectly predicts as negative.
- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- $Precison = \frac{TP}{TP+FP}$
- $\boxed{Recall = \frac{TP}{TP+FN}}$
- $F1\ Score = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$ ←needs attention

Our senario, find broken machine.
Don’t want FN!

Visualization

- Three Dimension at most —— three feature we made (more clearly)

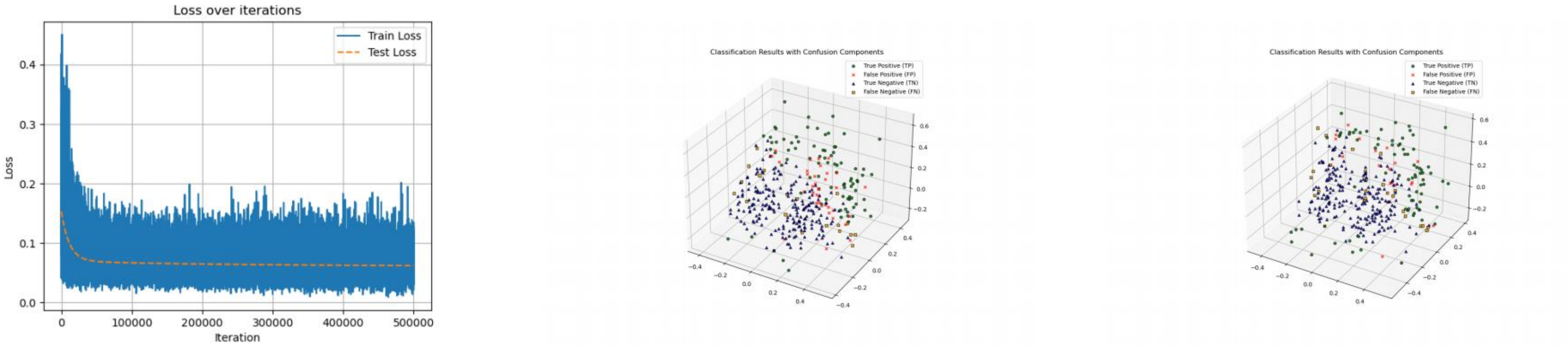


3

Experiment Result

Linear Regression

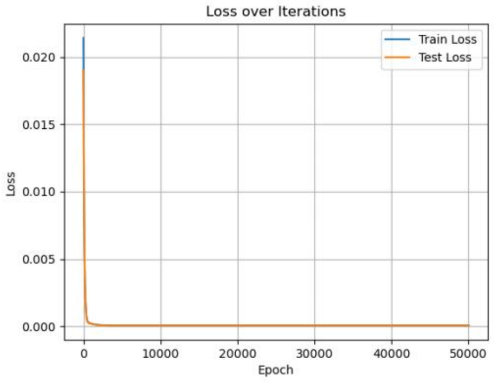
- TP: 83 TN: 206 FP: 36 FN: 21
- Accuracy: 0.8352601156069365 Precision: 0.6974789915966386 Recall: 0.7980769230769231 F1 Score: 0.7443946188340808
- Total time taken: 38.09750461578369 seconds



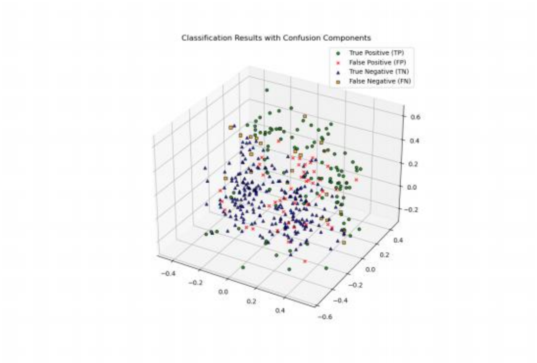
(a) Loss Trend (b) My Model Performance (c) scikit-learn Model Performance

Perceptron

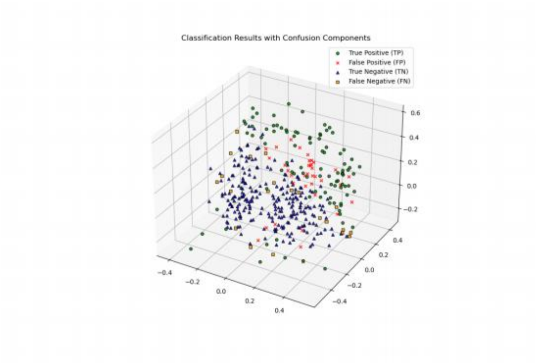
- TP: 102 TN: 242 FP: 44 FN: 21
- Accuracy: 0.8410757946210269 Precision: 0.6986301369863014 Recall: 0.8292682926829268 F1 Score: 0.758364312267658
- Total time taken: 19.484343767166138 seconds



(a) Loss Trend



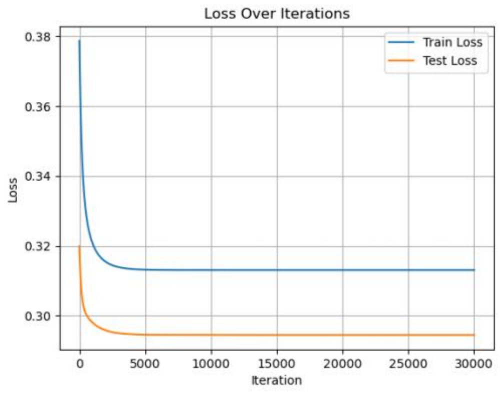
(b) My Model Performance



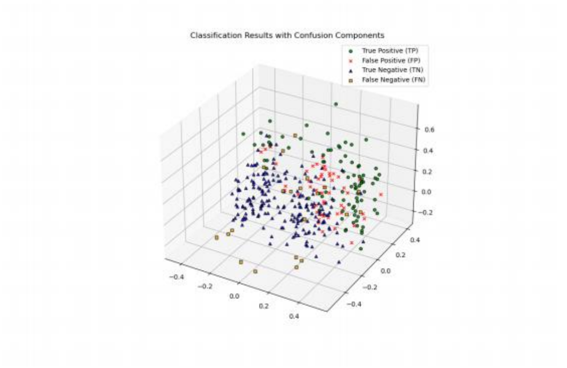
(c) scikit-learn Model Performance

Logistic Regression

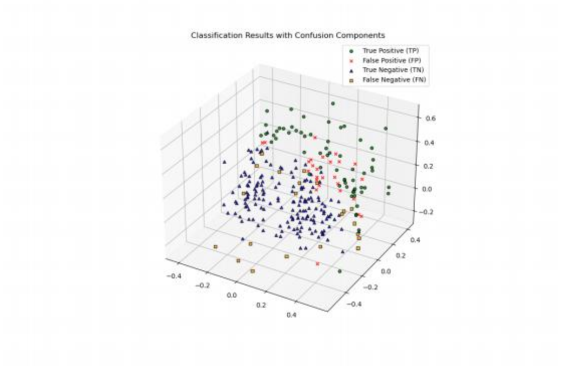
- TP: 85 TN: 195 FP: 52 FN: 21
- Accuracy: 0.7932011331444759 Precision: 0.6204379562043796 Recall: 0.8018867924528302 F1 Score: 0.6995884773662552
- Total time taken: 19.374540090560913 seconds



(a) Loss Trend



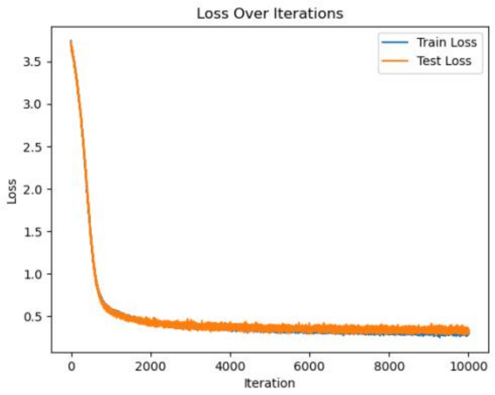
(b) My Model Performance



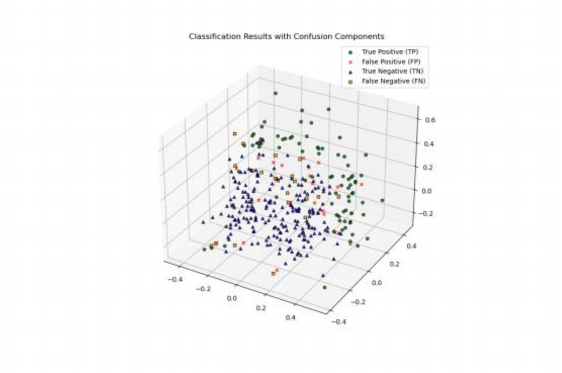
(c) scikit-learn Model Performance

Multi-Layer Perceptron

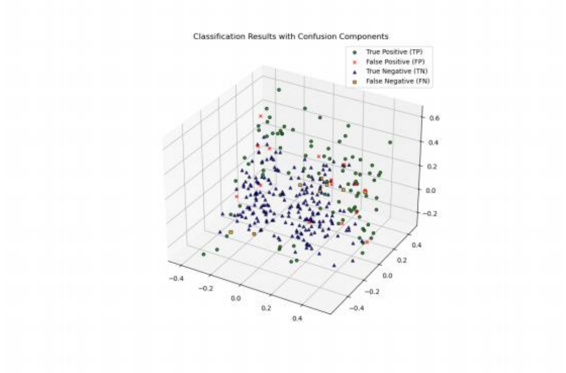
- TP: 87 TN: 213 FP: 20 FN: 13
- Accuracy: 0.9009009009009009 Precision: 0.8130841121495327 Recall: 0.87 F1 Score: 0.8405797101449274
- Total time taken: 421.797244310379 seconds



(a) Loss Trend



(b) My Model Performance



(c) scikit-learn Model Performance

Performance Summary

Model	TP	TN	FP	FN	Accuracy	Precision	Recall	F1-Score
Linear Regression	83	206	36	21	0.8353	0.6977	0.7980	0.7443
Perceptron	102	242	44	21	0.8411	0.6986	0.8292	0.7584
Logistic Regression	85	195	52	21	0.7932	0.6204	0.8019	0.6996
Multi-layer Perceptron	87	213	20	13	0.9009	0.8131	0.8700	0.8405

Analyzation

- **Linear Regression and Perceptron** —— linear model —— can’t work well with nonlinear problem
- **Logistic Regression** —— can’t use complex activation function, can’t go deep into features
- **Multi-Layer Perceptron** —— work best here, but slower, need difficult hyperparameter adjustion, while still not good enough

Advanced Try

- Look into the dataset’s paper!
- A. Complex Classifier Training and Performance
- After initial evaluation and optimization of support vector machines, artificial neural networks we settle for a **bagged trees** ensemble classifier. This is to a certain extend intuitive as the database’s rules for machine failure are a combination of thresholds in at least two features. The classifier’s performance is shown in Table I and can be considered satisfactory for our purpose.
- B. Explainable Model Training
- As an explainable model we train a set of 15 **decision trees** limited to a maximum of only 4 nodes for easy interpretability by a human. Each decision tree is trained using only 4 of 6 available features in the pattern shown in Table II . An example decision tree (number 1) is shown in Fig. 1 .

Matzka, S. (2020). Explainable Artificial Intelligence for Predictive Maintenance Applications. 2020 Third International Conference on Artificial Intelligence for Industries (AI4I), 69-74.

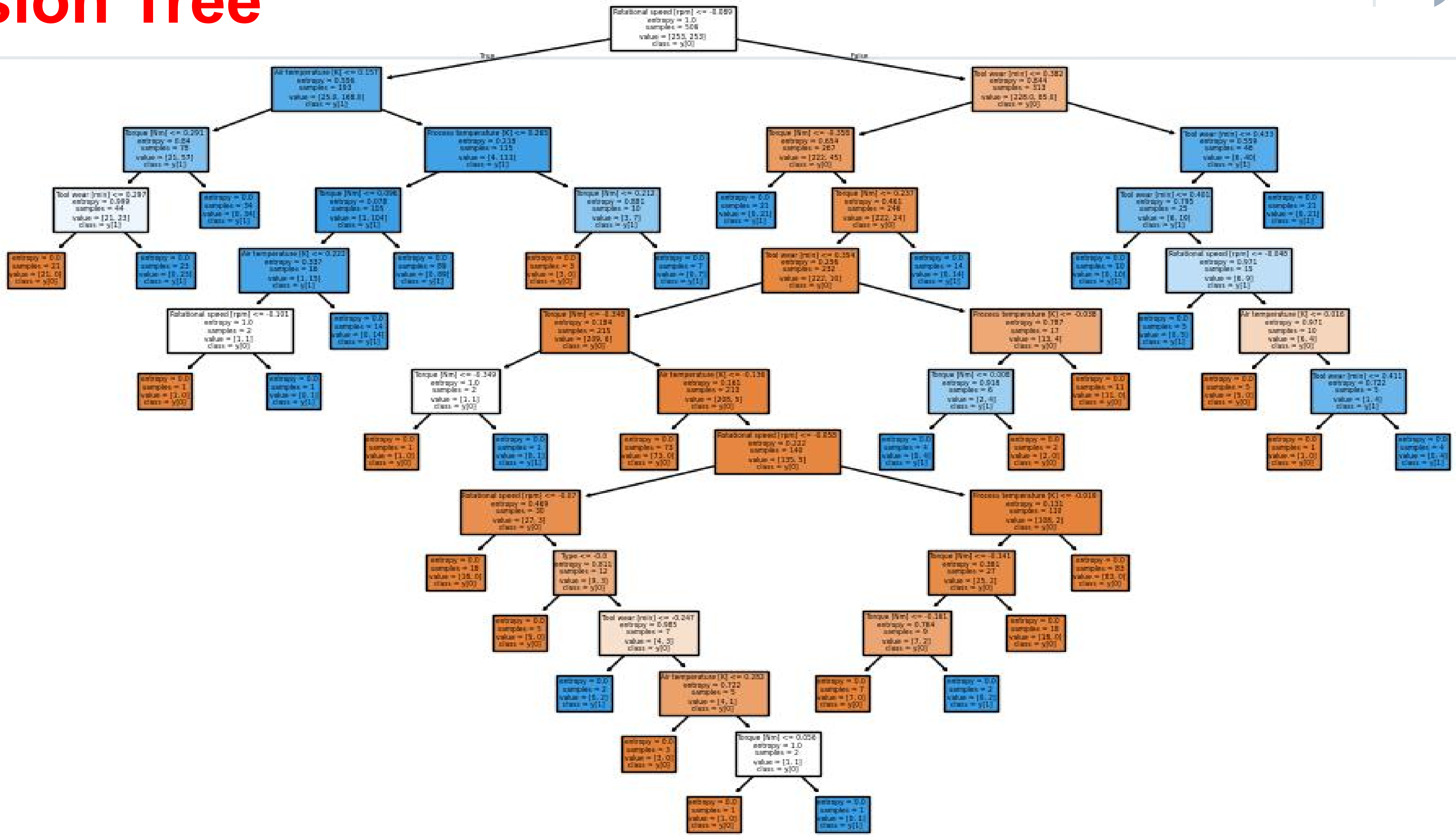
Advanced Try

- Realize by decision tree ourselves!

My Decision Tree		**true class**	
		failure	operation
predicted class	failure	77	271
	operation	9	2643

Author's Bagged Decision Tree		**true class**	
		failure	operation
predicted class	failure	294	45
	operation	121	9540

Decision Tree



4

Conclusion

Conclusion

- For linear model, hard to detect nonlinear relations.
- Data Preprocessing, Feature Engineering, really works well!
- Optimization on learning rate and update method helps with converge speed and performance.
- Multi-Layer Perceptron, with dynamic designs (lr, update, dropout) can work much better than without them.
- Such (feature1 and feature2) problem (known from priority knowledge) can work much better with decision tree.
- If used in real industry? More complex model, more hyperparamter adjustion, explanable methods……

Thanks

Ying Yiwen