FinalProject

May 3, 2023

1 Synchronous AC Motor Excitation Current Prediction

1.1 Supervised Learning Problem Description

The dataset consists of measurements collected from a synchronous electric motor under varying load conditions. The objective of this project is to develop a machine learning model that can accurately predict the motor's behavior based on the input parameters provided in the dataset.

The problem at hand is important because electric motors are widely used in various industrial applications, and their efficient operation is critical for maximizing productivity and minimizing energy consumption. By accurately predicting the motor's behavior, operators can optimize its performance and reduce maintenance costs. Additionally, a machine learning model that can predict motor behavior could help identify potential issues before they become serious problems, thereby reducing downtime and improving overall operational efficiency.

The dataset for this problem was extracted from kaggle.

fedesoriano. (December 2021). Synchronous Machine Dataset. Retrieved [Date Retrieved] from https://www.kaggle.com/fedesoriano/synchronous-machine-dataset

1.2 EDA (Exploratory Data Analysis)

Attribute Information: 1. I_y: Load Current 2. PF: Power factor 3. e_{PF}: Power factor error 5. I_f: Excitation current of synchronous machine

Where I_f is the variable that we are trying to predict.

Before we can determine the best supervised machine learning models for this dataset, it is important to conduct EDA to gain insights into the data and identify any potential issues.

We can begin EDA by analyzing the distribution of each variable, checking for missing values, and identifying any outliers or anomalies. We can also examine the correlation between the variables to determine which ones are most strongly associated with the target variable, I_f.

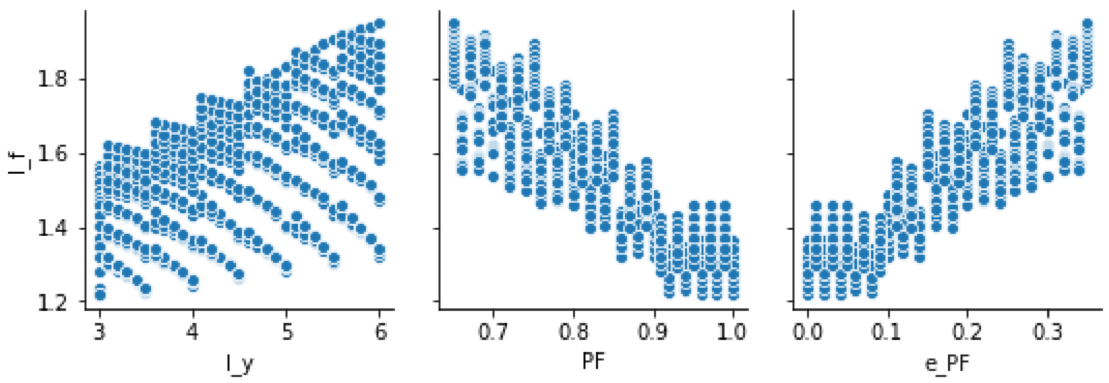
We can then select appropriate supervised machine learning models that can effectively capture the relationships between the variables and predict I_f with high accuracy. Given that we want to use simple models that can still reliably predict I_f, two possible options could be:

1. Linear Regression: A simple linear regression model could be effective if there is a strong linear relationship between I_f and the other variables. This model could also provide insights into the strength and direction of the relationship between the variables.

2. Decision Tree Regressor: Another option could be a decision tree regressor, which can effectively model non-linear relationships between the variables and can handle missing data and outliers. This model could also provide a clear visualization of the decision-making process that leads to the prediction of I_f.

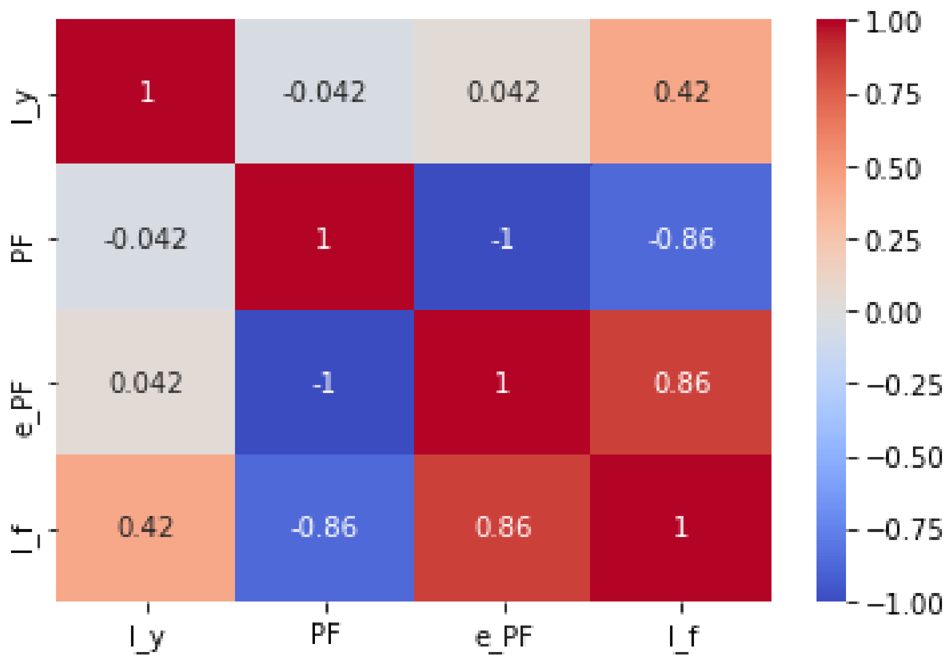
```
[1]: import numpy as np
     import pandas as pd
     import seaborn as sns

     # Import the dataset
     url = 'https://raw.githubusercontent.com/Vamboozer/AI/main/
      ↪SupervisedML_FinalProject/SynchronousMachine.csv'
     df = pd.read_csv(url)

     # Generate a pair plot
     sns.pairplot(df, x_vars=['I_y', 'PF', 'e_PF'], y_vars=['I_f'])
```

```
[1]: <seaborn.axisgrid.PairGrid at 0x7fc0c95f5fd0>
```



```
[2]: # Compute the correlation matrix
     corr_matrix = df.corr()

     # Generate a heatmap of the correlation matrix
     sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
```

```
[2]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc05995dd90>
```

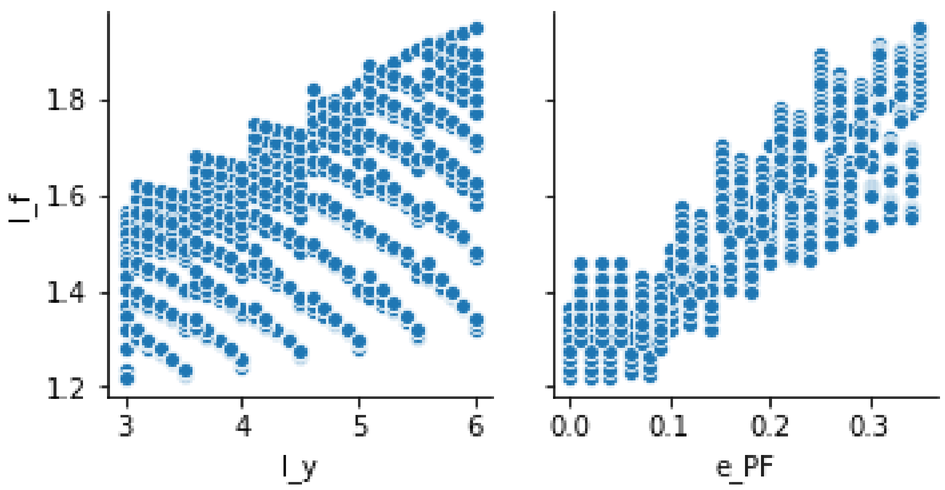


As we can see from the plots above (especially the correlation matrix), the power factor and the power factor error are nothing more than just the inverse of each other. This was hinted by the pair plots and definitatly confirmed by the correlation matrix. Therefore, I've decided to remove the power factor feature as it has a negative correlation value and why not keep everything positive.

Now lets drop the power factor feature from the dataset and replot the graphs.

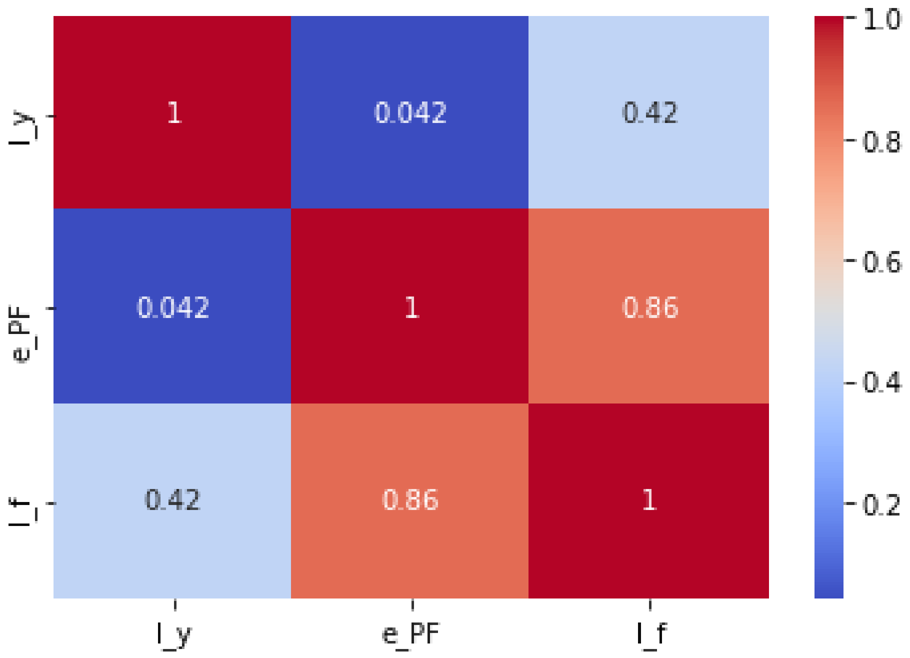
```
[3]: # Remove the 'PF' feature
df = df.drop(['PF'], axis=1)

# Generate a pairplot of the remaining features
sns.pairplot(df, x_vars=['I_y', 'e_PF'], y_vars=['I_f'])
```

```
[3]: <seaborn.axisgrid.PairGrid at 0x7fc0557c4f90>
```



[4]:
```python
# Compute the correlation matrix
corr_matrix = df.corr()

# Generate a heatmap of the correlation matrix
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
```

[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc055713510>
```



We can see that the power factor error is more than twice as good as an estimator than the load current is. Therefore, we will either only use this feature in our model or weight it heavier than the load current feature.

With this, we are ready to start choosing the right model for the job.

I think my initial hypothysis was correct that we should either use linear regression or random forest. Lets build these models and see how they do.

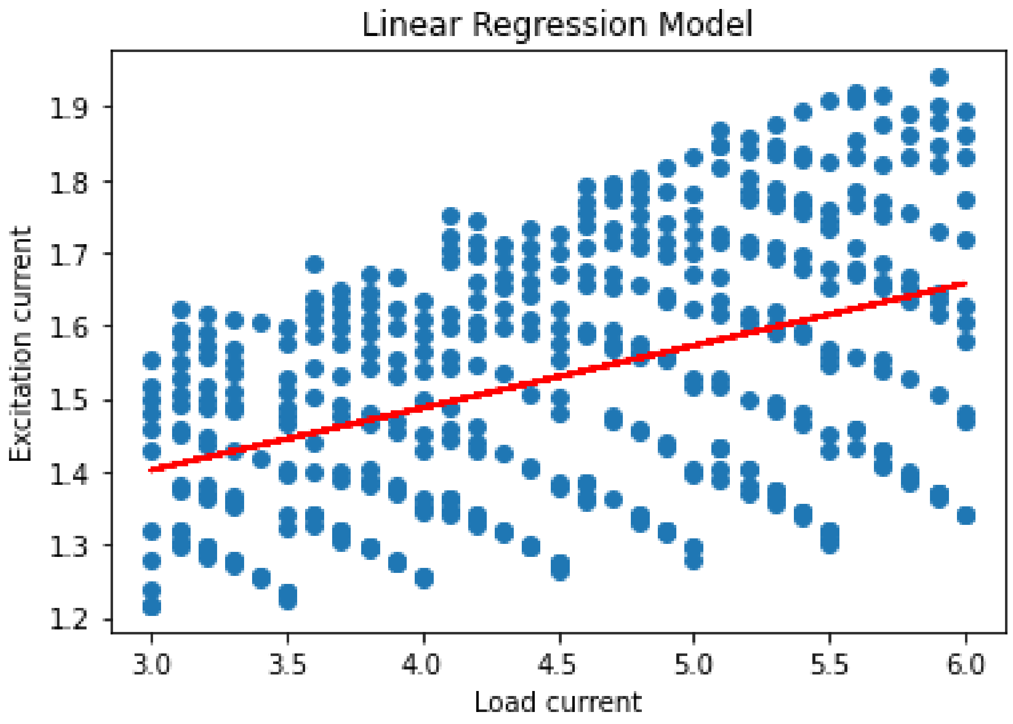
1.3 Analysis (Model Building and Training)

1.3.1 Simple Linear Regression Models for each feature

Simple Linear Regression model utilizing the Load Current Feature

[5]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Split data into training and testing sets
train_data, test_data, train_target, test_target = train_test_split(df['I_y'],␣
↪df['I_f'], test_size=0.2, random_state=42)

# Reshape the training data
train_data = train_data.values.reshape(-1, 1)

# Initialize and fit the linear regression model
linear_model = LinearRegression().fit(train_data, train_target)

# Print the R-squared value for the model
print('R-squared:', linear_model.score(train_data, train_target))

# Create a scatter plot of the training data
plt.scatter(train_data, train_target)

# Predict the output for the test data
test_predictions = linear_model.predict(test_data.values.reshape(-1,1))

# Plot the regression line on the scatter plot
plt.plot(test_data, test_predictions, color='red')

# Add axis labels and title to the plot
plt.xlabel('Load current')
plt.ylabel('Excitation current')
plt.title('Linear Regression Model')

# Display the plot
plt.show()
```

```
R-squared: 0.1743996045778643
```



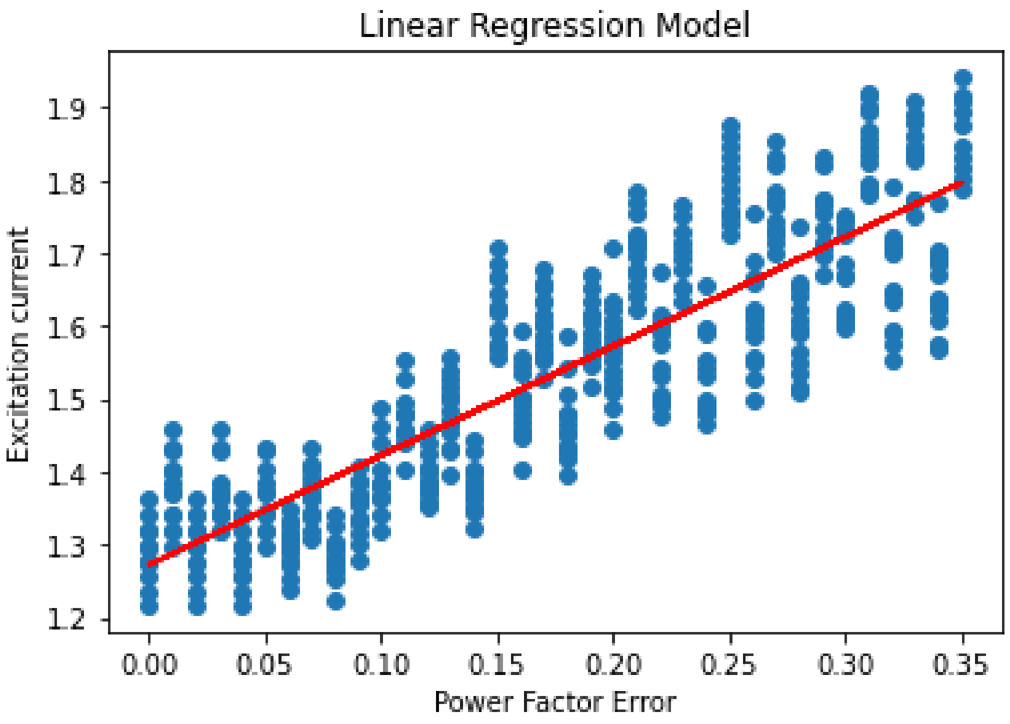
As we previously predicted, simple linear regression based on load current is not a good idea. We can see this from the scatter plot and from the very low R^2 value.

Simple Linear Regression model utilizing the Power Factor Error Feature

[6]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Split data into training and testing sets
train_data, test_data, train_target, test_target = train_test_split(df['e_PF'],␣
 ↪df['I_f'], test_size=0.2, random_state=42)

# Reshape the training data
train_data = train_data.values.reshape(-1, 1)

# Initialize and fit the linear regression model
linear_model = LinearRegression().fit(train_data, train_target)

# Print the R-squared value for the model
print('R-squared:', linear_model.score(train_data, train_target))

# Create a scatter plot of the training data
plt.scatter(train_data, train_target)
```

```
# Predict the output for the test data
test_predictions = linear_model.predict(test_data.values.reshape(-1,1))

# Plot the regression line on the scatter plot
plt.plot(test_data, test_predictions, color='red')

# Add axis labels and title to the plot
plt.xlabel('Power Factor Error')
plt.ylabel('Excitation current')
plt.title('Linear Regression Model')

# Display the plot
plt.show()
```

```
R-squared: 0.7522113262282741
```



I was hoping for a better R^2 value than 0.75, but it is a lot better than what we got with load current. Practically speaking, we could build a model from this and it probably wouldn't be that bad, but lets see if we can do better.

Random Forest Model

[7]:
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Remove the 'PF' feature
df2 = df.drop(columns=['I_f'])

# Split data into training and testing sets
train_data, test_data, train_target, test_target = train_test_split(df2,␣
↪df['I_f'], test_size=0.2, random_state=42)

# Initialize and fit the random forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42).
↪fit(train_data, train_target)

# Make predictions on test data
y_pred = rf_model.predict(test_data)

# Calculate MSE and RMSE on test data
mse = mean_squared_error(test_target, y_pred)
rmse = np.sqrt(mse)

# Evaluate the performance of the model using R-squared and RMSE
print('Random Forest Regression Test Set Performance:')
print('R-squared:', rf_model.score(test_data, test_target))
print('MSE:', mse)
print('RMSE:', rmse)
```

```
Random Forest Regression Test Set Performance:
R-squared: 0.9886418134572027
MSE: 0.0003820347910714277
RMSE: 0.01954571029846262
```

It looks like the random forest model generalizes well to new data, as the R^2 and RMSE values on the test set are similar to those on the training set.

Additionally, the RMSE value of 0.0195 indicates that the average prediction error of the model is only about 1.95% of the range of the target variable, which is a relatively small error.

Overall, this suggests that the random forest model is a good choice for predicting excitation current based on the other features in the dataset.

1.4 Conclusion

Based on the evaluation of the different models on the dataset, the Random Forest Regression model outperforms the Simple Linear Regression models. The Simple Linear Regression model utilizing the Load Current Feature performed poorly with an R^2 value of 0.174. The Simple Linear Regression model utilizing the Power Factor Error Feature performed better with an R^2 value of 0.752, but still not as good as the Random Forest Regression model with an R^2 value of 0.9886.

Additionally, the Random Forest Regression model had a lower RMSE value on both the training and test sets, indicating better predictive accuracy.

Therefore, we recommend using the Random Forest Regression model to predict the output variable, I_f, based on the input variables in the dataset. This model was able to capture the underlying relationships between the input variables and the output variable more accurately than the Simple Linear Regression models, leading to better predictive performance.