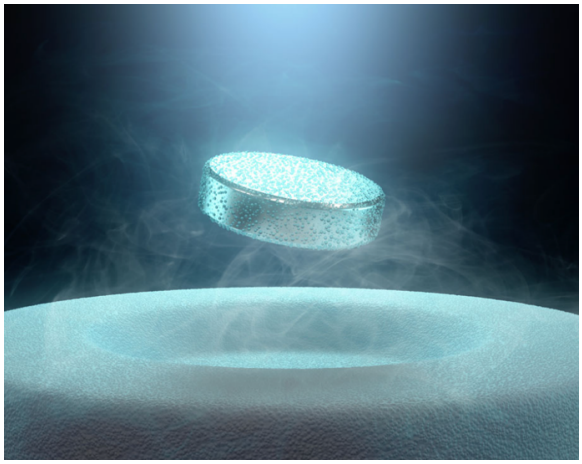
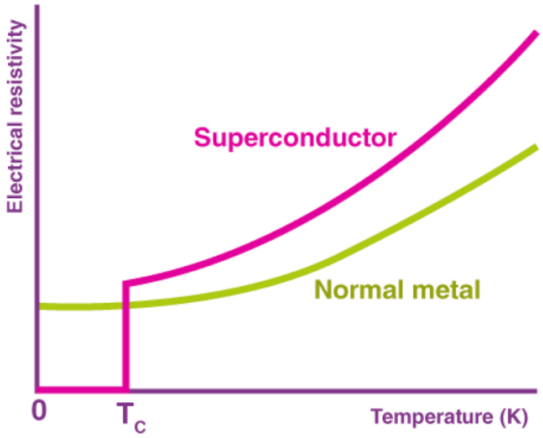
Prediction of Critical Temperature of Superconductors Using Polynomial Regression and MLP

Stage 2





INDEX

Introduction:

All materials can be classified as Conductors, Semiconductors and Insulators based on their resistance to the flow of electric charges. Superconductors are materials with zero resistance. Therefore, an electronic current can be maintained indefinitely without 'external voltage' [1]. However, superconductivity can only be maintained below a specific temperature called the critical temperature. Superconductors must be refrigerated to maintain superconductivity, for example, Liquid Helium (4.2 K) and Liquid Nitrogen (77 K) [2]. Researchers from all over the world have tried to fabricate superconductors that work at room temperature, but they have yet to succeed so far. A team of scientists from Korea University recently claimed that they had achieved superconductivity at room temperature in a compound known as LK-99 [3], a purple crystal made up of Lead, Copper, Phosphorous and Oxygen. However, those claims were met with scepticism as the attempts to replicate the results by other researchers worldwide failed to show superconductivity at room temperature, and the critical property of levitation was attributed to magnetism instead [4].

Based and inspired on the above recent news, the report aims to use the Machine Learning methods of Polynomial Regression and MLP (with mean squared error) to predict the critical temperature of the superconductors. The application domain is Material Physics. For Stage 2, the outline of this report begins with the introduction, which leads to the problem formulation and methods used section, which are then analysed in the results and conclusion section. The code is provided as an appendix. The report also includes cover page and references.

Problem Formulation:

The task is to predict the critical temperature of superconductors, framing this as a regression problem. The dataset was sourced from Kaggle from the Superconductivity-data dataset repository [5], wherein each data point represents a unique superconductor and its associated chemical properties. The dataset comprises 21,263 such data points and is characterised by a multidimensional input, with 81 feature values detailing various chemical properties of the superconductors, such as Number of elements, Atomic Mass, Atomic Radius, Density, Electron Affinity, Fusion Heat, Thermal Conductivity, Valence, and each of their statistical measures such as Mean, Standard Mean, Weighted Mean, Entropy, and Range, making it a multi-dimensional dataset. The features are continuous numerical values representing the various chemical properties and their statistical measures. The target label for this problem is the critical temperature, provided in the 82nd column of the dataset, a continuous numerical value indicating the temperature in Kelvin (K) below which a material exhibits zero electrical resistance. The types of data present in the dataset are primarily continuous numerical values detailing the various chemical attributes and their statistical inferences.

Methods:

The dataset, sourced from the Superconductivity-data dataset repository on Kaggle [5], comprises 21,263 data points, each representing a unique superconductor and its associated chemical properties. The dataset has 81 features characterising chemical properties and their statistical inferences. Critical temperature is the label.

Due to the different scales of the feature values, pre-processing involved standardising the features to have zero mean and unit variance. This is essential in using PCA and any regression task due to the scales' sensitivity. For feature selection process, PCA was employed to reduce the dataset's dimensionality while retaining most of its variance. This helped simplify the model by retaining the data's most informative aspects eliminating redundant features. Setting the n_components parameter to 0.95 in PCA ensures the selection of principal components that are the most significant features, in terms of variance (95%), are retained.

Two machine learning models were put to use:-

Polynomial Regression- This model was chosen for its simplicity and interpretability. A polynomial regression with a degree of 3 was used to capture non-linear relationships between the features and the target variable. The linear hypothesis space allows for understanding relationships in datasets with many features.

Multi-layer Perceptron (MLP)- An “MLP regressor” was chosen to capture complex, non-linear relationships in the data. The choice of MLP was motivated by its ability to model patterns using its hidden layers, making it suitable for high-dimensional datasets.

The Mean Squared Error loss function was used to understand the models. MSE gives the average squared difference between the actual and predicted values and directly measures the average error in predictions for evaluating the quality of the hypothesis in regression problems.

The dataset was split into training, validation, and test sets. The initial split allocated 70% of the data to the training set and 30% to a temporary set. This temporary set was split equally into validation and test sets, ensuring a 70-15-15 split. The size of each set are, training set with 14,884 data points, validation set and test set with 3,190 data points each. This design choice ensures sufficient data for training while still having sizable validation and test sets for model evaluation. This approach seems best for predicting critical temperature.

Results:

Comparison of Training and Validation Errors for the 2 ML methods chosen:

Polynomial Regression

Training Error: 189.9848478612272 K
Validation Error: 228.51029470094286 K

MLP

Training Error: 10.38788031712691 K
Validation Error: 11.275905532423021 K

Upon comparing the training and validation errors, it is highly evident that the MLP model significantly outperforms the Polynomial Regression model. The MLP model achieved a much lower error on the training and validation sets, indicating its superior predictive capability for this dataset.

Therefore, The Multi-layer Perceptron (MLP) is the final chosen method. The decision is based on its lower validation error than the Polynomial Regression model. The reason is the lower error indicates that the MLP model has a better generalisation capability and is more adept at capturing the underlying patterns in the data.

For the MLP model, we had created the test set as 15% of the dataset, the test error turns out to be 12.1797589440262 K. This error provides an unbiased estimate of the model's performance on unseen data, further solidifying the choice of MLP as the prime model for this task.

Conclusion:

This report was based on predicting the critical temperature of superconductors using machine learning. We found MLP to be the most effective in predicting the label through pre-processing, feature selection via PCA, and comparatively evaluating two distinct models. The Polynomial Regression model, although simpler, did not perform as well as the MLP. It is found that training and validation errors by the Polynomial Regression model are way higher than MLP. This suggests that the relationships in the data might be more complex than what a third-degree polynomial can tell. For MLP model, the training error was around just 1 K less than the validation error which is great. The test error for the MLP then turned out to be around 12.2 K.

Even though the MLP model turned to be highly optimal, improvements could involve using neural networks and deep learning. Another improvement could be incorporating more recent data, which is hard to find publicly.

Prediction of Critical Temperature of Superconductors using ML methods has been done before [6]; however, using different ML Methods like Clustering, Random Forests, etc., the Use of Polynomial Regression and MLP has been uncommon and has yet to be made.

References:

[1] Wikipedia page on Superconductivity: https://en.wikipedia.org/wiki/Superconductivity#

[2] OCU Superconductivity Lab: https://classes.oc.edu/PhysicsLab/SUPER_C.htm#:~:text=There%20are%20several%20advantages%20in,cold%20than%20does%20liquid%20helium

[3] The controversial Arxiv paper: https://arxiv.org/abs/2307.12008

[4] Replication attempts table: https://en.wikipedia.org/wiki/LK-99#Experimental_studies

[5] Dataset source, Kaggle: https://www.kaggle.com/datasets/tunguz/superconductivty-data-data-set

[6] The same concept using the Random Forest method: https://www.nature.com/articles/s41524-018-0085-8

Appendix:

Project

October 7, 2023

[10]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from nose.tools import assert_equal
from numpy.testing import assert_array_equal
import seaborn as sns
%config Completer.use_jedi = False
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPRegressor
```

[3]:
```
data = pd.read_csv('trainsupercon.csv')

# Split the dataset into training, validation, and test sets
train_data, temp_data = train_test_split(data, test_size=0.3, random_state=42)
val_data, test_data = train_test_split(temp_data, test_size=0.5,␣
 ↪random_state=42)

# Features and Labels
X_train = train_data.drop('critical_temp', axis=1)
y_train = train_data['critical_temp']
X_val = val_data.drop('critical_temp', axis=1)
y_val = val_data['critical_temp']
X_test = test_data.drop('critical_temp', axis=1)
y_test = test_data['critical_temp']

# Standardizing the data here is important because the numeric data contains␣
 ↪datapoints with different scales
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

[4]:
```
# Applying PCA
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_val_pca = pca.transform(X_val)
X_test_pca = pca.transform(X_test)
# Reason of choosing n_components = 0.95: we need to build principal components␣
 ↪where we have a very high variance as that
# will allow me to find features that have better range of values so that I get␣
 ↪better results. For every prediction, therefore I will
# look for high variance. This is because I am doing feature importance.
```

[5]:
```
# Polynomial Regression

# model
poly = PolynomialFeatures(degree=3)
    # Not running a loop to find degree with lowest errors because dealing with␣
 ↪lots of data that
    # lead to runtime error and feel degree 3 will be the best choice after␣
 ↪checking them obviously

# Training
lin_regr = LinearRegression(fit_intercept=False)
X_train_polyreg = poly.fit_transform(X_train_pca)
lin_regr.fit(X_train_polyreg, y_train)
X_val_polyreg = poly.transform(X_val_pca)
X_test_polyreg = poly.transform(X_test_pca)
# Prediction
y_pred_polyreg_train = lin_regr.predict(X_train_polyreg)
y_pred_polyreg_val = lin_regr.predict(X_val_polyreg)
y_pred_polyreg_test = lin_regr.predict(X_test_polyreg)
# Error
polyreg_tr_error = mean_squared_error(y_train, y_pred_polyreg_train)
polyreg_val_error = mean_squared_error(y_val, y_pred_polyreg_val)
polyreg_test_error = mean_squared_error(y_test, y_pred_polyreg_test)
print(f"Training Error: {polyreg_tr_error}, Validation Error:␣
 ↪{polyreg_val_error}, Test Error: {polyreg_test_error}")
```

```
Training Error: 189.98484786122708, Validation Error: 228.51029470094528, Test
Error: 246.82813288521538
```

[6]:
```
# MLP

# Model
mlp = MLPRegressor(hidden_layer_sizes=(100, 50), max_iter=1000,␣
 ↪random_state=42) # No need for loop as we just want one instance because of␣
 ↪large data
```

```
# Training
mlp.fit(X_train_pca, y_train)
# Predictions
y_train_pred_mlp = mlp.predict(X_train_pca)
y_val_pred_mlp = mlp.predict(X_val_pca)
y_test_pred_mlp = mlp.predict(X_test_pca)
# Errors
train_error_mlp = mean_squared_error(y_train, y_train_pred_mlp, squared=False)
val_error_mlp = mean_squared_error(y_val, y_val_pred_mlp, squared=False)
test_error_mlp = mean_squared_error(y_test, y_test_pred_mlp, squared=False)
print(f'Training error: {train_error_mlp}, Validation error: {val_error_mlp},␣
 ↪Test Error: {test_error_mlp}')
```

```
Training error: 10.387880317126953, Validation error: 11.275905532423137, Test
Error: 12.17975894402637
```

[7]:
```
printer = """
The errors of the two machine learning models to predict the critical␣
 ↪temperature of superconductors, Polynomial Regression and MLP:

• Polynomial Regression-
    Training Error: {} K
    Validation Error: {} K
    Test Error: {} K

• MLP-
    Training Error: {} K
    Validation Error: {} K
    Test Error: {} K

The MLP model very significantly outperforms the Polynomial Regression model.␣
 ↪MLP becomes the final primary model for this task.

""".format(polyreg_tr_error, polyreg_val_error, polyreg_test_error,␣
 ↪train_error_mlp, val_error_mlp, test_error_mlp)

print(printer)
```

```
The errors of the two machine learning models to predict the critical
temperature of superconductors, Polynomial Regression and MLP:

• Polynomial Regression-
    Training Error: 189.98484786122708 K
    Validation Error: 228.51029470094528 K
    Test Error: 246.82813288521538 K
```

- MLP-
  Training Error: 10.387880317126953 K
  Validation Error: 11.275905532423137 K
  Test Error: 12.17975894402637 K

The MLP model very significantly outperforms the Polynomial Regression model. MLP becomes the final primary model for this task.