Machine learning approaches for the component optimization of airfoils

Abstract

High aircraft fuel consumption is a driving force behind global carbon emissions. Emissions in the aviation industry are expected to increase, potentially comprising a quarter of the 1.5 °C carbon budget by 2050. Improving the efficiency of airfoil design is key to the creation of more fuel-efficient airplanes. Unfortunately, current component optimization is inaccessible due to its reliance on computational fluid dynamics, which is expensive, difficult to use, and computationally expensive, resulting in inefficient designs. Rapid advancements in machine learning (ML) techniques open new avenues to make this process faster and more affordable. We constructed a series of generative ML models and evaluated their simulation runtimes and accuracy. Each model was trained to output the structure of an airfoil that most closely produces desired lift, drag and moment coefficients. Models utilizing ensemble methods, including Random Forest Regressors, Extra Trees Regressors, Bagging Regressors and K-Nearest Neighbors Bagging, as well as the multilayer perceptron, produce accurate and efficient predictions of airfoil geometries. Machine learning models present a significant improvement in runtime and ease of use over traditional computational fluid dynamic software.

Keywords: Airfoil, Computational Fluid Dynamics, Generative Design, Machine Learning

Introduction

Background Information

An airfoil is a two-dimensional cross-sectional geometry that is profiled in a way which generates lift, drag, and rotational moment as fluids pass around it. A few parameters can be used to describe the general structure of an airfoil, including camber, camber position, thickness, thickness position and alpha value (angle of attack), as shown in Figure 1.

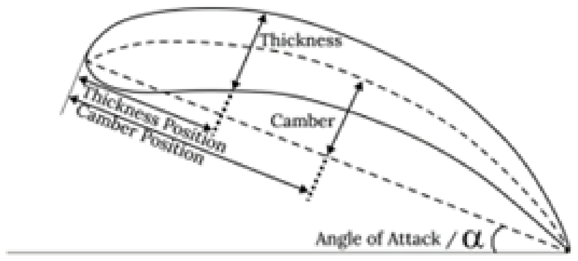

Figure 1. Diagram of an airfoil and its related parameters.

In recent years, Computational Fluid Dynamics (CFD) has matured into a versatile, pervasive tool that can accurately simulate the behavior of fluid flow [1]. Despite substantial improvements made in processor performance, iterative CFD, a technique often used in component optimization, is computationally expensive and relatively inaccessible. Due to the complexity of the governing equations, high dimensionality of the design space, and high time resolution needed to evaluate turbulence, it may require hundreds of core hours to simulate the performance of one geometry iteration. This challenge exponentially increases as the scale of the model increases [2]-[3]. Thus, CFD problems are often solved at national supercomputer centers or similar facilities with limited accessibility [4]. Due to a lack of access, engineers may make design decisions based on insufficient data, potentially leading to performance deficits, including inefficient energy consumption. In pursuit of environmental sustainability, researchers have concentrated on exploring more accessible tools to optimize airfoil design. Hauser et al. at the University of Kentucky attempted to create lower cost computing facilities in the early 2000s. They clustered multiple workstations and servers, which reduced computational cost compared to more powerful computational models [4].

Since then, advancements in machine learning (ML) have opened new avenues to more efficient alternatives to CFD. It is possible to exploit ML models' ability to discern patterns in large datasets with high dimensionality. Many studies have explored the use of ML models as a replacement to CFD, both generally and for specific applications (e.g., wind flow in built environments) [5]-[8]. Some studies, such as Peng et al. and Weiwei et al., applied models to airfoil design problems [9]-[11]. These studies attempted to create models for the airflow and aerodynamic coefficients of specified airfoils with drastically reduced runtime.

Problem Description

Currently, camber, camber position, thickness and thickness position can be entered into software, such as XFLR5, to find the airfoil's lift, drag and moment coefficients. For simulating more complex problems relating to airfoils, one must use computational fluid dynamics software that is (a) difficult to obtain for the average person, (b) expensive, (c) often requires the use of powerful computers, and (d) requires a steep learning curve. Additionally, machine learning models in the literature have been designed to predict the aerodynamic coefficients of airfoils, not their geometries. There is yet to be an accessible generative design approach for the topological optimization of an airfoil. We propose the creation of a machine learning model as an alternative to CFD software, which will take as input the desired coefficients of the airfoil and output a possible structure. A machine learning model would be easy to distribute (e.g. via web app), could be free of cost, does not require a powerful computer and demands less technical ability to use. This will allow engineers to design airfoils that will conform to specific coefficients of drag and lift, hopefully increasing the fuel efficiency of aircraft. To accomplish this goal, we constructed five distinct machine learning models and compared the resulting runtimes and accuracy.

Methods

Data Acquisition and Preparation

The data used for training and validation consists of nine attributes per airfoil: airfoil name, thickness, thickness position, camber, camber position, alpha, lift coefficient, drag coefficient and moment coefficient. The following steps describe how these values were obtained for the final training dataset of 101,445 airfoils. All code used to acquire and prepare the dataset was custom-made in Python.

The raw airfoil coordinates were scraped from two online databases: the University of Illinois at Urbana-Champaign Airfoil Coordinates Database and the Airfoil Coordinate Database by the Technical University of Delft. A script automatically entered all the coordinate files into an airfoil analysis software named XFOIL, producing a polar file for each downloaded airfoil. Another script copied the values from each polar file, compiling them all together in one CSV file. The values in the dataset vary depending on the alpha value, incrementing from 0 to 30 in steps of 0.5. All simulations performed by XFOIL used the lifting-line theory to approximate the polar performance at identical conditions (Reynolds number = 10000, NCrit = 9.0, Mach number = 0, max number of iterations = 500, and the fluid properties are based on air at atmospheric pressure). See Figure 2 for a visualization of the data pre-processing.

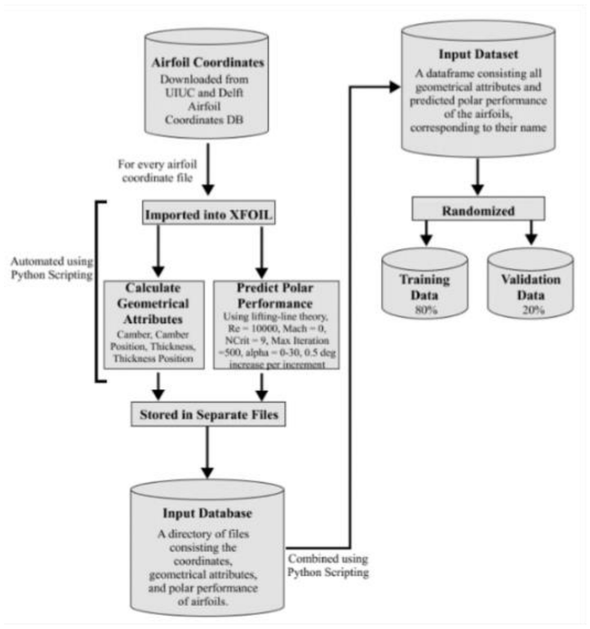
Airfoil Coordinates
Downloaded from UIUC and Delft Airfoil Coordinates DB
For every airfoil coordinate file
Imported into XFOIL
Automated using Python Scripting
Calculate Geometrical Attributes
Camber, Camber Position, Thickness, Thickness Position
Predict Polar Performance
Using lifting-line theory, Re = 10000, Mach = 0, NCrit = 9, Max Iteration =500, alpha = 0-30, 0.5 deg increase per increment
Stored in Separate Files
Input Database
A directory of files consisting the coordinates, geometrical attributes, and polar performance of airfoils.
Combined using Python Scripting
Input Dataset
A dataframe consisting all geometrical attributes and predicted polar performance of the airfoils, corresponding to their name
Randomized
Training Data
80%
Validation Data
20%

Figure 2. Flowchart demonstrating data preparation and feature selection.

All clearly invalid data (negative values for geometric properties and empty cells) were eliminated manually. The entire dataset was randomized, split into features (lift, drag and moment coefficients) and targets (camber, camber position, thickness, thickness position, alpha), and finally split – 80% for training and 20% for validation.

Model Construction

We built five machine learning models, each of which takes as input the lift, drag and moment coefficients, and outputs the geometrical attributes (thickness, thickness position, camber, camber position and alpha) of a corresponding airfoil. All models were built in Python using a variety of machine learning libraries. See Figure 3 for the general flow of training and prediction for the models.

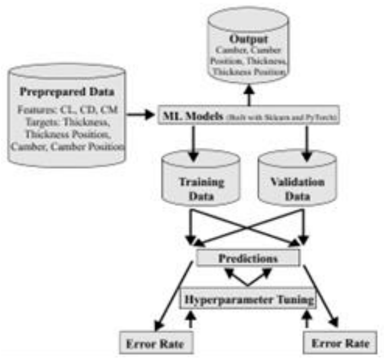
Output
Camber, Camber Position, Thickness, Thickness Position
Preprepared Data
Features: CL, CD, CM
Targets: Thickness, Thickness Position, Camber, Camber Position
ML Models (Built with Sklearn and PyTorch)
Training Data
Validation Data
Predictions
Hyperparameter Tuning
Error Rate
Error Rate

Figure 3. General Model Training and Prediction Flowchart.

The first model was a multilayer perceptron (MLP), a simple neural network composed of layers of nodes that modifies and feeds inputs forwards until they reach the output layer [12].

The model was built with 5 layers overall, using PyTorch, and was trained for 500 epochs with a batch size of 32 per epoch.

Second, we built a K-Nearest Neighbors (KNN) model using SciKit-Learn, which learns continuous data by assigning labels to query points computed based on the mean values of its k nearest neighbors [13] We utilized a grid search and brute testing to find the optimal values of k, and subsequently trained the KNN.

Third, we created a set of ensemble models, which combined the predictions of several base estimators trained from a given algorithm to generalize data and predict accordingly [14]. These were a Random Forest Regressor (RF), Extremely Random Trees Regressor (ExtraTrees) and Bagging Regressor, all of which were constructed using Sci-Kit Learn.

Hyperparameter Optimization

Hyperparameters for the RF, ExtraTrees, Bagging Regressor and KNN models were tuned to optimize their accuracy. We optimized the number of base estimators for the ensemble models (RF, ExtraTrees and Bagging Regressor). For the KNN, we optimized the constant *k* number of nearest neighbors that should be evaluated when predicting a value.

We graphed the number of base estimators against both runtime and average residual error to calculate the optimal number of base estimators. For each incrementation of the number of base estimators, each model was trained and tested (i.e., predicted data). During testing, the accuracy and runtime was recorded. See Figure 4 and 5 for the resulting runtime and residual error per number of base estimators.

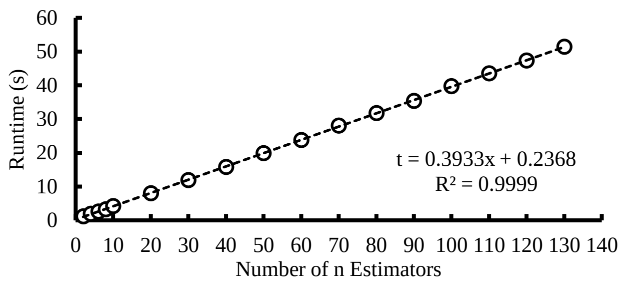

Figure 4. Runtime of an Increasing Number of n Estimators for an RF Regressor.

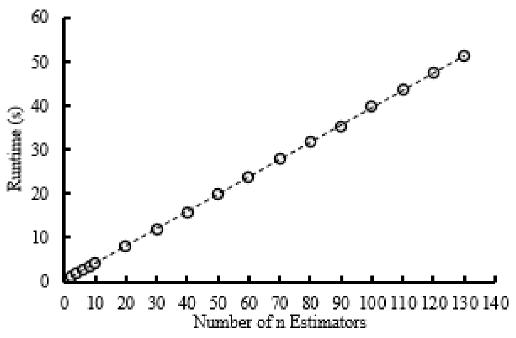

Figure 5. Mean Squared Error of an Increasing Number of n Estimators for an RF Regressor.

We determined the optimal number of base estimators to be 35 for the RF model, 50 for ExtraTrees and 35 for Bagging. This is the number of base estimators at which the average residual error begins to asymptote, and therefore will improve very little if the number of base estimators is increased any further. The lowest value possible was chosen to minimize runtime.

For the KNN, we used two competing methods to find the value of k that produces the least error. First, we iterated through *k = 0* to *k = 41*, fitting the KNN regressor each time and calculating the relative standard error (See Figure 6). The result is that approximately a value of 2 for *k* would be least error prone.

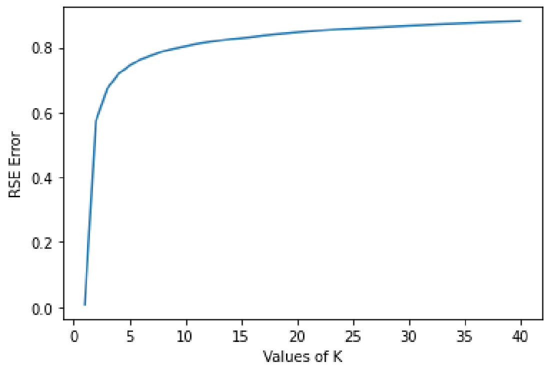

Figure 6. RSE for Each K Neighbors.

Second, we conducted a grid search of all values *k* up to 200. In the same grid search we calculated whether the weights should be uniform or distance. The results were a value of *k = 113* and uniform weights. Finally, we used bagging (averaging the predicted values across many instances of KNNs) to attempt to further reduce error (while keeping *k = 113*).

Performance Metrics for Familiar Airfoils

We define “familiar” airfoils as any airfoil that was not used to train the model but was still obtained from the same original databases. “Unseen” airfoils are those we created to test the limits of the ML models.

We used two metrics to assess the performance of each model when predicting “familiar” airfoils. The first was the runtime of both training and validating the model in Google Colab, which has a built-in function that records the total runtime of each block of code to the nearest one-thousandth of a second. The runtime does not include time spent preparing the data because this must only be done once before the models are in use. The second was the average residual error during the validation phase. Since average residual error is dependent on the value being measured, we only used this metric to compare models with each other, and not to CFD software.

Results

Acceptable ranges of average residual error are dependent on the value being measured. It may seem like the alpha values had much higher error than camber, but this is simply because camber values are usually in the range of 0 to 0.1, whereas the alpha values in the training dataset ranged from 0 to 30. Therefore, the absolute error for alpha is much larger than that of camber, even though they may have similar percentage error.

The multilayer perceptron produced airfoil structures similar to the validation dataset structures. Some of the average residual errors are quite low, such as 0.6014 for alpha, which has a median value of 15, and 0.0291 for thickness, which typically ranges from 0.1 to 0.4. The model's runtime was long, at 157 seconds for training and validation. The results are shown in Table 1.

Table 1. Total and Average Residual Error for the Multilayer Perceptron.

Regressors	**Total Residual Error**	**Average Residual Error**	**Runtime (s)**
Thickness	778.57	0.0291	157.15
Thickness Position	1353.88	0.0506	
Camber	345.05	0.0129	
Camber Position	3023.62	0.1129	
Alpha	16105.64	0.6014	

Overall, the multilayer perceptron was the least accurate and second longest running model. The K-Nearest Neighbors model was more accurate and generally had a lower runtime, as shown in Table 2. We built three models according to two different ways of optimizing the model's principal hyperparameter k and an additional bagging model, where the final result was the mean of the predictions from 100 KNNs. Of the three models, the bagging KNN with *k = 113* generated the lowest error and highest runtime.

Table 2. Total and Average residual error for each configuration of the KNN.

		K		
Regressor	**Accuracy Metric**	**2**	**113**	**113 + Bagging**
Thickness	Total residual error[JF1]	623.78	582.40	571.22
	Average Residual Error	0.03074	0.02871	0.02815
Thickness Position	Total Residual Error	1155.08	1014.23	1008.39
	Average Residual Error	0.05693	0.04999	0.04970
Camber	Total Residual Error	244.09	221.88	219.13
	Average Residual Error	0.01203	0.01094	0.01080
Camber Position	Total Residual Error	2427.46	2176.23	2128.25
	Average Residual Error	0.1196	0.1073	0.1049
Alpha	Total Residual Error	12731.50	12711.64	12570.31
	Average Residual Error	0.6275	0.6265	0.6196
All	Runtime (s)	25.127	26.094	619.615

All three models utilizing ensemble methods – RF regressor, ExtraTrees, and Bagging regressor – generated promising results. The average runtimes were short, ranging from approximately 3 to 15 seconds. However, predicted runtime for usage (solely prediction) would be even lower. The models were able to predict thousands of iterations of airfoils in a matter of seconds. Overall, ExtraTrees recorded the shortest runtime, at 3 to 5 seconds. ExtraTrees may have had lower runtime compared to RF, even though it is essentially a randomized version of RF, because it had a lower number of n estimators. A larger number of n estimators likely increased the runtime of the RF and Bagging regressors. While the three models had similar average residual error, ExtraTrees was the most accurate model.

The average residual error for the RF regressor, ExtraTrees and Bagging Regressor are shown in Table 3. The percentage error for the ExtraTrees model is approximately 10-15% across all output values.

Table 3. Residual error and runtime for the RF, ExtraTrees and bagging models.

Regressor	**Accuracy Metric**	**Random Forest**	**ExtraTrees**	**Bagging**
Thickness	Total Residual Error	527.464	503.315	560.035
	Average Residual Error	0.02600	0.02481	0.02760
	Runtime (s)	11.205	3.555	16.179
Thickness Position	Total Residual Error	966.454	948.934	987.370
	Average Residual Error	0.04763	0.04677	0.04866
	Runtime (s)	9.720	2.761	13.365
Camber	Total Residual Error	207.045	206.250	211.054
	Average Residual Error	0.01020	0.01017	0.01040
	Runtime (s)	9.455	2.944	12.017
Camber Position	Total Residual Error	2076.219	2043.077	2101.257
	Average Residual Error	0.102333	0.10069	0.10356
	Runtime (s)	10.821	2.869	13.347
Alpha	Total Residual Error	11421.459	11804.183	11253.457
	Average Residual Error	0.56291	0.58177	0.55463
	Runtime (s)	8.317	2.810	10.452

The model with both the lowest average residual error and runtime is the ExtraTrees regressor. It has approximately 10-15% error for each of the five outputs and a runtime of 3-5 seconds. Error and runtime are given as ranges because they were calculated from one subset of airfoils from the validation dataset. Therefore, there may be larger variations in error and runtime if the models were to be tested against airfoils outside of the UIUC and TU Delft airfoil databases.

Discussion

The machine learning models constructed in this investigation relatively accurately predict the structure of new airfoils according to desired lift, drag and moment coefficients. Of the five models, the ensemble methods show the lowest average residual error and runtime, predicting airfoil geometries accurately to within 10-15% error. They demonstrate a significant reduction in runtime and computational cost in comparison to traditional CFD software. Whereas CFD software running on a typical household computer may take hours to simulate thousands of airfoils, the ExtraTrees model takes seconds. Furthermore, existing CFD software predicts the lift, drag and moment coefficients from the geometry of different airfoils, whereas our models do the opposite: design airfoils according to the desired coefficients. Thus, our models streamline the process of "reverse engineering" airfoils from desired specifications.

A key limitation of the models created is that they do not output a definitive set of airfoil coordinates, and instead only descriptive geometric attributes. From this arise two issues. First, the user would need to design their own airfoil from the geometry. However, this would not be overly complex and in fact grants the user more flexibility to conform to certain design specifications. Second, the model in its current state acts more as a highly specific search filter, rather than a self-sufficient generative design model. In reality, a set of lift, drag and moment coefficients may correspond to multiple airfoil structures.

Another limitation is the relatively small number of features in the dataset, which reduces the models' ability to identify other correlations between airfoils. This is partially why some of the models have residuals that substantially affect its output airfoil structures. Given that the aerodynamics of a component are very sensitive to change, the mean residual error can harm the performance of predicted airfoils. In our attempt to reduce the dimensionality of the model in order to reduce computational cost, we ignored many factors impacting the performance of airfoils, such as the boundary layer condition and surface roughness.

The training dataset has airfoils with alpha values up to 30 degrees. However, it is extremely unlikely an aircraft ever flies at such an angle of attack. This skewed the predicted airfoils towards higher alpha values, which is unrealistic to implement.

Conclusion

We constructed five machine learning models that produced the geometric structure of airfoils according to input lift and drag coefficients. The ensemble methods produce the most accurate airfoils with the lowest runtimes. They may allow engineers to design airfoils according to desired specifications much faster than conventional computational fluid dynamic software. Considering that emissions from the aviation industry are expected to comprise a quarter of the 1.5 °C carbon budget by 2050, increasing the fuel efficiency of planes is of utmost importance [15]. The models presented can be used to (a) save time and money in the process of designing aircraft, and (b) be used as a tool to design more fuel-efficient aircraft by inputting certain drag and lift coefficients.

Acknowledgements

We used the following Python libraries: Numpy 1.21.5, Pandas 1.3.5, Matplotlib 3.2.2, TensorFlow 2.8.0, IPython 5.5.0, SciPy 1.4.1, Seaborn 0.11.2, Torch 1.10.0, and Scikit Learn 1.0.2. Training data is publicly available in the UIUC and TU Delft Airfoil Coordinate Databases. The following software was used: XFOIL, XFLR5, and Google Colab.

References

[1] Norton T, Sun D-W. Computational Fluid Dynamics (CFD) – an effective and efficient design and analysis tool for the Food Industry: A Review. Trends in Food Science & Technology. 2006;17(11):600–20.

[2] Jeon J, Lee J, Kim SJ. Finite Volume Method Network for the acceleration of unsteady computational fluid dynamics: Non-reacting and reacting flows. International Journal of Energy Research. 2022;46(8):10770–95.

[3] Chen W, Chiu K, Fuge M. Aerodynamic design optimization and shape exploration using generative adversarial networks. AIAA Scitech 2019 Forum.

[4] Hauser T, Mattox TI, LeBeau RP, Dietz HG, Huang PG. High-cost CFD on a low-cost cluster. ACM/IEEE SC 2000 Conference (SC'00).

[5] Usman A, Rafiq M, Saeed M, Nauman A, Almqvist A, Liwicki M. Machine Learning Computational Fluid Dynamics. 2021 Swedish Artificial Intelligence Society Workshop (SAIS).

[6] Vinuesa R, Brunton SL. The potential of machine learning to enhance computational fluid dynamics [Internet]. arXiv.org. 2021 [cited 2022Aug6]. Available from: https://arxiv.org/abs/2110.02085v1.

[7] Kochkov D, Smith JA, Alieva A, Wang Q, Brenner MP, Hoyer S. Machine learning–accelerated computational fluid dynamics. Proceedings of the National Academy of Sciences. 2021;118(21).

[8] Calzolari G, Liu W. Deep learning to replace, improve, or aid CFD analysis in Built environment applications: A review. Building and Environment. 2021;206:108315.

[9] Peng W, Zhang Y, Desmarais M. Deep Neural Network for airfoil optimization. AIAA SCITECH 2022 Forum.

[10] Zhu L, Zhang W, Kou J, Liu Y. Machine learning methods for turbulence modeling in subsonic flows around airfoils. Physics of Fluids. 2019;31(1):015105.

[11] Wu H, Liu X, An W, Chen S, Lyu H. A deep learning approach for efficiently and accurately evaluating the flow field of supercritical airfoils. Computers & Fluids. 2020;198:104393.

[12] Menzies T, Kocagüneli E, Minku L, Peters F, Turhan B. Using goals in model-based reasoning. Sharing Data and Models in Software Engineering. 2015;:321–53.

[13] 1.6. nearest neighbors [Internet]. scikit. [cited 2022Aug6]. Available from: https://scikit-learn.org/stable/modules/neighbors.html#regression

[14] 1.11. ensemble methods [Internet]. scikit. [cited 2022Aug6]. Available from: https://scikit-learn.org/stable/modules/ensemble.html#ensemble

[15] Pidcock R, Yeo S. Analysis: Aviation could consume a quarter of 1.5C carbon budget by 2050 [Internet]. Carbon Brief. 2016 [cited 2022Aug6]. Available from: https://www.carbonbrief.org/aviation-consume-quarter-carbon-budget