Applying Machine Learning Approaches for Understanding Turbulent Flow

Reza Momenifar & Long Zhang

Abstract

Using numerical experiments data of a turbulent flow field, different machine learning models including linear regression, K nearest neighbors and multilayer perceptron are employed to build a data-driven model that relates the velocity gradients of the field to its pressure Hessian. The results show that the model with the highest performance can capture up to 80% of variability in the test set which is a promising achievement due to the chaotic nature of turbulence.

1 Problem Description

In the context of fluid dynamics, turbulent flow is viewed as a chaotic, rotating, multi-scale flow which is very difficult to analyze (due to non-linearity and non-locality) and in fact in classical physics, turbulence is regarded the most difficult unsolved problem ((Eames and Flor, 2011)). Kinetic energy of turbulent flows, after nonlinear interactions between different scales of flow, eventually is dissipated into internal energy by the fluid's molecular viscosity at small scales.

A long standing goal of turbulence modeling has been to understand the underlying physics of the small-scales in turbulent flow (Chevillard et al. (2008)), particularly via modeling the velocity gradient tensor [1], $\boldsymbol{A} \in \mathbb{R}^{3\times3}$, which gives comprehensive insight into the dynamical, statistical and topological properties of small scales motions in turbulence. This tensor has components defined by $A_{ij} = \boldsymbol{\nabla u} = \partial u_i/\partial x_j$, in which u_i is a component of the fluid velocity, and x_i is a spatial coordinate. The velocity field can be computed by solving the following incompressible Navier-Stokes equations for a specific domain and boundary conditions:

$$\frac{\partial}{\partial t}\boldsymbol{u} + \boldsymbol{u}\cdot\boldsymbol{\nabla u} = -\frac{1}{\rho}\boldsymbol{\nabla} p + \nu\boldsymbol{\nabla}^2\boldsymbol{u} \tag{1}$$

where t is time, ρ is density, p is pressure and ν is the kinematic viscosity ($\boldsymbol{\nabla}^2$ is the Laplacian operator). Taking the spatial gradient of the NS equations yields:

$$\frac{\partial}{\partial t}\boldsymbol{A} + \boldsymbol{u}\cdot\boldsymbol{\nabla A} = -\boldsymbol{A}^2 - \boldsymbol{H} + \nu\boldsymbol{\nabla}^2\boldsymbol{A} \tag{2}$$

in which $\boldsymbol{H}$ is the Hessian of the pressure field, with components $H_{ij} = \frac{\partial^2 p}{\partial x_i \partial x_j}$. This equation is unclosed because it requires information from the non-local (anisotropic) part of pressure Hessian and viscous term. One can find several efforts in previous studies to remedy this problem by neglecting these terms using different approaches including Restricted Euler (RE approximation) (such as Cantwell (1992)), stochastic dynamical model (such as Chevillard and Meneveau (2007)) which lead to finite time singularities and unrealistic results at high Reynolds number [2], respectively (Chevillard et al. (2008)). Motivated by a number of recent studies (Johnson and Meneveau (2016); Lawson and Dawson (2015); Wilczek and Meneveau (2014)) in exploring the closure models for pressure Hessian and viscous Laplacian in equation 2, the focus of this project is on developing a model for the pressure Hessian term, aiming to improve our understanding of small scale turbulent motions.

Formally, the pressure Hessian depends on a spatial integral of the invariants of the tensor $\boldsymbol{A}$ (Chevillard et al. (2011)). This spatial integral is complicated and difficult to analyze, and a long standing challenge of turbulence modeling research has been to develop a simplified model that relates $\boldsymbol{H}$ to the invariants of $\boldsymbol{A}$. The two highly relevant invariants of velocity gradient tensor (A) are $Q = -\frac{1}{2}tr(\boldsymbol{A}^2)$ and $R = -\frac{1}{2}tr(\boldsymbol{A}^3)$ (known as "(R, Q)-plane"). The simplified model for pressure Hessian closure should provide a reasonable approximation to $\boldsymbol{H}$, yet be simple enough to be amenable to analysis. The hypothesis of this project is that given (large amounts of) data for the tensor fields

[1] A first-order tensor is a vector and has three components (3^1). A second-order tensor has nine components (3^2) and can be represented as a matrix.

[2] It is a non-dimensional number representing the intensity of a turbulent flow.

$\boldsymbol{H}$, $\boldsymbol{A}$ and its invariants, machine learning algorithms can be used to extract a (data-driven algorithm for) relationship between them. Here, the problem is treated as a supervised learning, in which a multi-target regression model is needed that takes velocity gradients information as input features and predicts the 6 components of $\boldsymbol{H}$.

2 Data Description

Our in-house code (Ireland et al. (2013)) solves the three-dimensional Direct Numerical Simulation (DNS) of equation 1 on a triperiodic cube of length $\mathscr{L}$, using a pseudo-spectral method on a uniform mesh with N^3 grid points (where N denotes number of grid points in each direction), and generates velocity field (at different time steps) in the spectral space corresponding to all the grid points in the physical space. For this project, the plan is to take velocity data of our simplest case, which has been computed (using 16 cores) on a cubic domain with $\mathscr{L} = 2\pi$, $N = 128$, and construct $\boldsymbol{A}$, and $\boldsymbol{H}$ components along with R and Q at each grid point ($128^3 = 2097152$ grid points per time step). It is worth mentioning that the pressure Hessian is a symmetric tensor and therefore $\boldsymbol{H}$ has 6 independent/unknown components. Furthermore, due to incompressibility condition, sum of the diagonal components of $\boldsymbol{A}$ are zero ($A_{ii} = 0$) (8 independent components) and so taking the trace of equation 2 results in $Tr(\boldsymbol{H}) = -Tr(\boldsymbol{A}^2)$.

Firstly we need to construct $A_{ij} = \partial u_i/\partial x_j$, knowing the fact that $A_{ij} = \mathscr{F}^{-1}[i\kappa_j \widehat{u_i}]$ [3]. To this aim, the velocity field $\widehat{u_i}$ ($\widehat{}$denotes Fourier space) in Fourier space is multiplied by the wavenumber (κ) and then is converted into physical space. Once $\boldsymbol{A}$ is obtained, $\boldsymbol{A}^2$ and $\boldsymbol{A}^3$ and eventually R and Q (invariants of $\boldsymbol{A}$) are computed.

Secondly, we need to obtain the Hessian of the pressure field, knowing that $H_{ij} = \frac{\partial^2 p}{\partial x_i \partial x_j} = \mathscr{F}^{-1}[-i\kappa_m\kappa_n \frac{i\kappa_j G_j}{\kappa^2}]$. To this goal, having velocity field $\widehat{u_i}$, u_i (velocity field in physical space) is computed in order to recover $u_i u_j$. Then, this tensor is converted back to Fourier space to compute $\widehat{u_i u_j}$. Afterwards, $G_j = i\kappa_1(\widehat{u_j u_1}) + i\kappa_2(\widehat{u_j u_2}) + i\kappa_3(\widehat{u_j u_3})$ is computed and finally converting $[-i\kappa_m\kappa_n \frac{i\kappa_j G_j}{\kappa^2}]$ to physical space, outputs Hessian of the pressure.

It is worth mentioning that the current results satisfy the $Tr(\boldsymbol{A}) = 0$ and $Tr(\boldsymbol{H}) = -Tr(\boldsymbol{A}^2)$ equations which confirms that the computed data are consistent with the underlying physics and hence are reliable.

3 Preprocessing

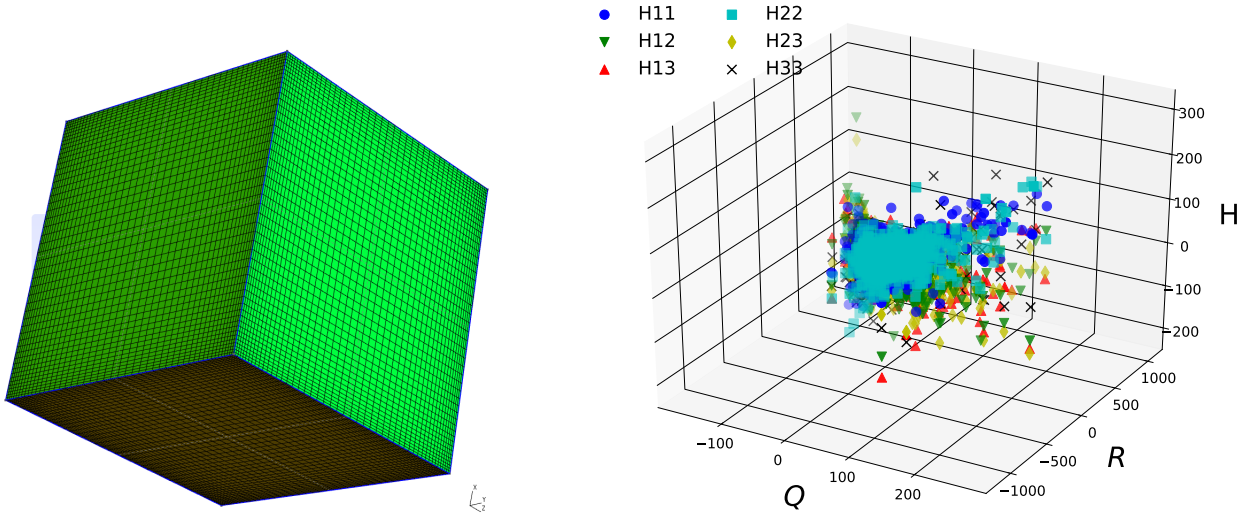
The data of this study, which has been extracted from the computational domain of the numerical solution shown in the figure 1a, has been printed out in the HDF5 (Hierarchical Data Format) format in 16 files (corresponding to 16 cores used to generate the velocity field data). Each file contains datasets of gird point indices (each grid is specified by three indices, each varying from 1 to 128), 9 components of $\boldsymbol{A}$ (A_{ij}, where $i, j = 1, 2, 3$), R, Q and 6 components of $\boldsymbol{H}$ (which are H_{11}, H_{12}, H_{13}, H_{22}, H_{23}, H_{33}). These files are read in consecutively so that three dimensional arrays of R, Q, components of $\boldsymbol{A}$ and components of $\boldsymbol{H}$ are ready to be processed. It should be mentioned that the focus of this project is not on time series data and so only datasets of one time step will be processed.

There are $128^3 = 2,097,152$ grid points and so in total $17 * 128^3 = 35,651,584$ datapoints are available ($9 * 128^3$ corresponding to the 9 components of $\boldsymbol{A}$, $2 * 128^3$ corresponding to R and Q as input features and $6 * 128^3$ outputs). The goal is to use all these datapoints (which can be considered a large dataset) to build the model. As a first step, 10 percent of dataponits (with random selection) are reserved as a test set and the rest are considered as training and validation sets. In order to build the input features, three different scenarios are compared. In the first scenario, R and Q are considered as the input features, the second one takes 9 components of $\boldsymbol{A}$ and the third one is their combination which has 11 features. The figure 1b visualizes a random sample of the dataset (around 1% of the grid points) based on the first scenario.

At the very first step, the data is split into the training and validation sets, where the former is employed to learn a model and the latter is used to evaluate the model performance and tune the model parameters. According to the summary statistics of the data, the input features have

[3] (inverse) Fourier transform notation. Given a function $f(t)$, its Fourier transform is $g(\omega) = \mathscr{F}[f(t)] = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(t) exp(-i\omega t)\, dt$ and the inverse Fourier transform is $f(t) = \mathscr{F}^{-1}[g(\omega)] = \int_{-\infty}^{\infty} g(\omega) exp(i\omega t)\, d\omega$.

a large and different variances and this characteristic indicates that the data need to be scaled (indeed, our preliminary results confirmed that the scaled data yields higher performance). Here, the input feature are standardized using the "StandardScaler" in the Scikit-learn library [4], in which subtracts off the mean and scales the variance to one, given that trying other scaling approaches such as MinMaxScaler and MaxAbsScaler did not improve the (preliminary) results.



(a) Computational domain

(b) Dataset visualization

Figure 1: a: Schematic of a cubic box with 128^3 uniform grid points. b: Scatter plot of the dataset (1% random sample of data) corresponding to the case 1 in which $\boldsymbol{H}$ components are plotted versus R and Q.

4 Methods

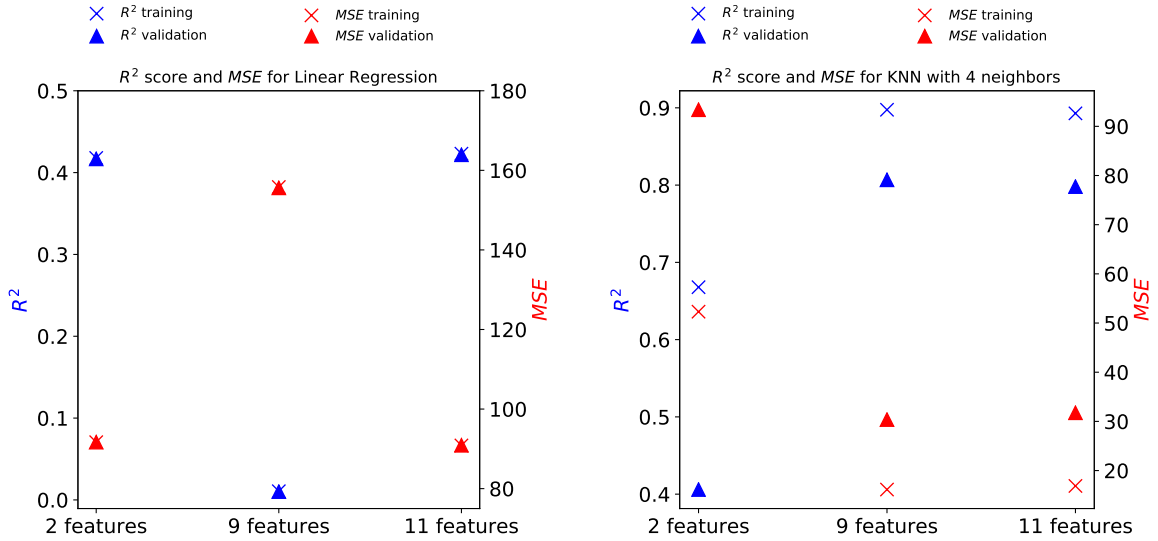
For this project, the Linear Regression (LR), the K Nearest Neighbors (KNN) and the multilayer perceptron (MLP) methods have been tested, for all the three different input cases. In the following sentences, these models are briefly introduced. In the linear regression approach, a model based on the linear combination of input features is fitted to the data so that the sum of the squared differences between the true targets and predicted ones is minimized (Indeed, it attempts to best minimize $RSS = \min_{\boldsymbol{\beta}} \|\boldsymbol{X}\boldsymbol{\beta} - y\|_2^2$, where $\boldsymbol{X}, \boldsymbol{\beta}$ and y represent inputs, model coefficients, and true targets, respectively). Despite the simplicity of the linear regression approach, it provides an insight into the extent to which a linear model can predict the variability within non-linear datasets of this project. In the KNN, a model is trained based on a number of training points (known as number of neighbors, K) which have the minimum distance (the default is the standard Euclidean distance) from a query point and the target of this new point is predicted via the mean of these nearby K points' targets. The optimum number of neighbors can be achieved by varying K in the KNN model and pick the one which results in the highest performance on the validation set. The MLP is a class of neural networks algorithms which can build complex models and is capable of learning the non-linear patterns (especially for large datasets). This algorithm can be characterized as the generalization of the linear models which involves multiple intermediate linear layers (known as hidden layers) with nonlinear activation function between the input and output layers. The complexity of a MLP model can be controlled by a number of hyperparameters such as number of the hidden layers, number of units (neurons) in each layer, the activation function, and the regularization, to name a few (Müller et al. (2016)). It should also be emphasized that neither the LR nor the KNN algorithms could capture the correlations between the outputs in a multi-target regression task and in fact they generate individual models corresponding to each target, while these correlations are considered in the MLP algorithm.

[4] is a free software machine learning library for the Python programming language

For all the aforementioned methods, the R^2 score and the mean squared error (MSE) metrics are employed to evaluate their performance [5]. The R^2 score metric, known as the coefficient of determination, ranges from 1 to $-\infty$ and explains the proportion of the total variation (of targets) captured by the model. The MSE metric assesses the average of squared errors and varies from 0 to $+\infty$. Therefore, the goal is to find a model with a higher R^2 and a lower MSE.

5 Results

Figure 2 illustrates the results of the performance evaluation of the linear regression and the KNN models on the both training and validation sets for the three scenarios of input features. According to the figure 2a, the performance of the linear regression on the both training and validation sets are nearly equal which implies that this linear model tend to underfit and so is unable to capture the underlying non-linear pattern of the data. Furthermore, one can see that while building a linear model with 2 or 11 input features can explain 42% of variability in the data, the model with 9 features (components of $\boldsymbol{A}$) results in a very poor performance. This behavior might indicate that only the R and Q features are important in predicting the $\boldsymbol{H}$ but this should be interpreted cautiously since the overall performance of the linear regression model on the non-linear datasets of this project is not promising (and consequently such a conclusion is not reliable). It should be noted that the linear regression model is scale invariant and so scaling does not affect the results.



(a) the Linear Regression Results

(b) the KNN Best Results

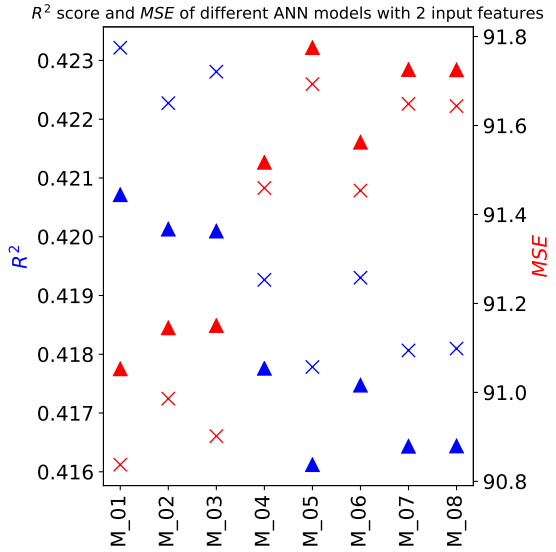
Figure 2: Evaluating the performance of the linear regression and the KNN models on both the training and validation sets, using the R^2 score and MSE metrics. Cross and triangle markers represent results of the training and validation sets, respectively. Blue color denotes the R^2 scores (left axis) and red color indicates the MSE magnitudes (right axis).

For the KNN model, the results of all the three cases show that scaling the input data can improve the performance around 25% while increasing the number of neighbors would decrease the performance on the validation data set. Here, only the results of the performance achieved with the optimum number of neighbors are shown. It is worth mentioning that although the best performance on the validation set has been achieved with only one neighbor, the optimum number of neighbors is chosen to be 4. The rational of this decision is that large and small differences between the performance of a model on training and validation sets indicate overfitting and underfitting, respectively and hence the optimum model should strike the balance between the two. From the results of the KNN with 4 neighbors in the figure 2b, one can see the model is away from overfitting

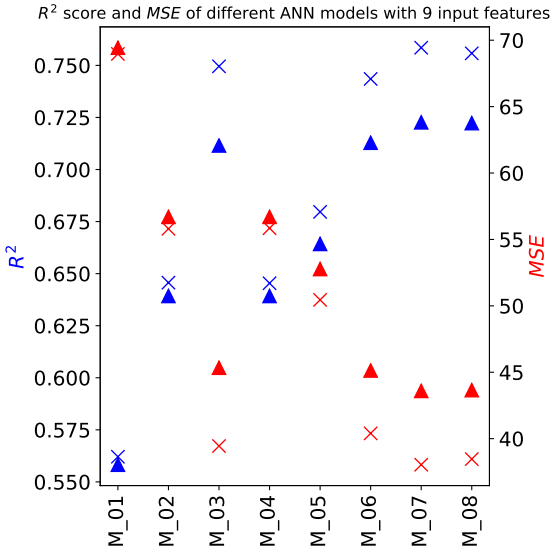
[5] $MSE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=1}^{n_{samples}} (y_i - \hat{y}_i)^2$, and $R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n_{\text{samples}}} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n_{\text{samples}}} (y_i - \bar{y})^2}$, where $\hat{y}$ is the predicted outcomes and y and $\bar{y}$ are the true outcomes and its mean, respectively.

and underfitting for all the cases and outputs promising performance. This figure reveals that adding features can enhance the performance (on the validation set) steeply (shifting R^2 score from 40% to 81% and reducing MSE from 92 to 19). Furthermore, the results show that unlike the linear regression model, the model based on 9 features performs twice better than the one with 2 features.

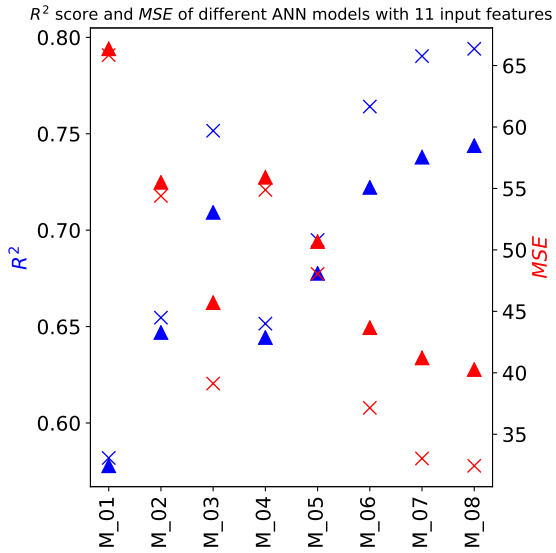
Turning now to the MLP results which are presented in the figure 3. The number of hidden layers and number of neurons per layer are the most important parameters in a MLP architecture (Müller et al. (2016)). Here for each input case, 8 different architecture of the MLP have been tested using the $NeuralNetwork.MLPRegressor$ package of the Scikit-learn library with 'Adam' as the optimization algorithm (setting tolerance to 10^{-6}), 'tanh' as the activation function and selecting a large number for the maximum iteration (1000) to ensure a converged solution. The architecture of hidden layers of different MLP models are as follows: $Model\,1 = [100]$, $Model\,2 = [1000]$, $Model\,3 = [1000, 500]$, $Model\,4 = [3000]$, $Model\,5 = [1000, 1000, 500]$, $Model\,6 = [1000, 1000]$, $Model\,7 = [1000, 1500]$, $Model\,8 = [1000, 2500]$.



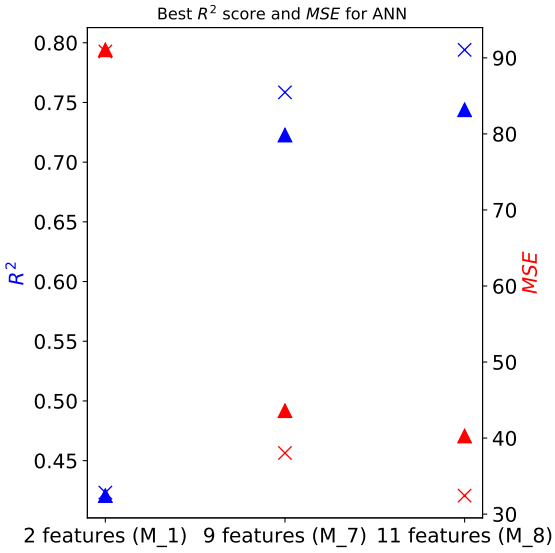
(a) 8 different MLP architecture with 2 inputs



(b) 8 different MLP architecture with 9 inputs



(c) 8 different MLP architecture with 11 inputs



(d) the Best MLP Results for the 3 input cases

Figure 3: Evaluating the performance of 8 different MLP models on both the training and validation sets, using the R^2 score and MSE metrics. The markers and their colors indicate the same meaning as the Figure 2 .

Figure 3a shows that using more complex architectures of the MLP could not improve the performance when the MLP models are trained with only two input features, while in the case of 9 (figure 3b) or 11 (figure 3c) inputs features, the gradual improvement of the performance by using different hidden layers architecture becomes clear and in fact the R^2 scores (on the validation set) shift to 72% and 74.5%, respectively. Furthermore, the figure 3d describes that the MLP with 11 features outperforms the 9 features' case, which was not apparent in the KNN model.

Finally the performance of the best achieved models on the test set (figure 4) indicates that the results are almost the same as validation set and this characteristic implies that the models generalize well to the unseen data.

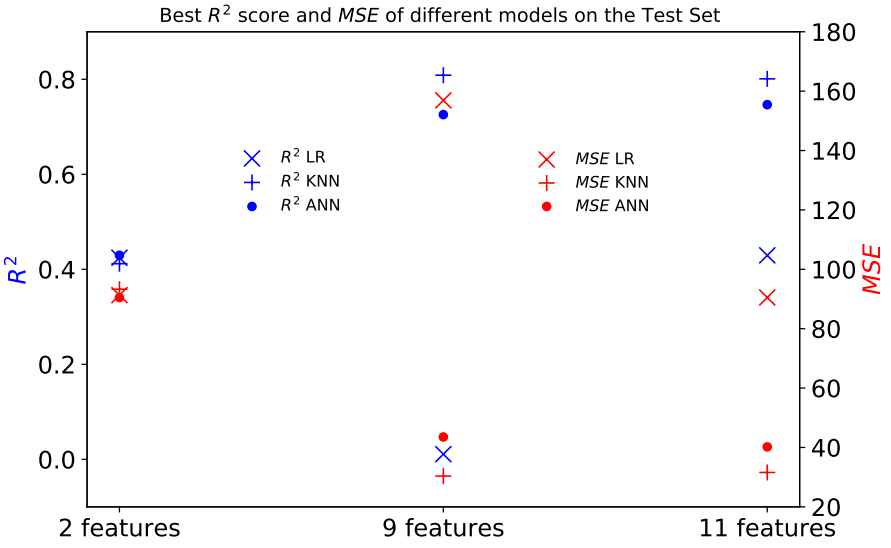


Figure 4: Evaluating the best performance of the LR, the KNN, and the MLP models, generated for the 3 input scenarios, on the test set, using the R^2 score and MSE metrics. Cross, plus and triangle markers represent results of the LR, the KNN and ANN, respectively. Blue color denotes the R^2 scores (left axis) and red color indicates the MSE magnitudes (right axis)

6 Conclusions

In this project, multiple models using linear regression, the KNN and the MLP algorithms were tested to build an efficient model that relates the velocity gradients of a turbulent flow field to its pressure Hessian. Due to the non-linear nature of the data (and in general turbulent flow field), a linear model fails to perform well, while the KNN shows the capacity to grasp the underlying chaotic pattern and yields the R^2 score of 80% which is considered as a decent performance for our purpose. It should be emphasized that unlike the MLP, none of these two approaches could capture the correlations between the outputs and in fact they generate 6 individual models, corresponding to the 6 components of $\boldsymbol{H}$. Although the performance of the MLP ($R^2 = 75\%$) was lower than the KNN, it showed that adding more input features can improve the performance. Future research should consider the potential effects of enriching the model by feeding more relevant features (such as components of $\boldsymbol{A^2}$ or other invariants of $\boldsymbol{A}$) more carefully and see how much it improves the performance. This, of course, requires substantial computational powers and seeking ways to accelerate the computations, such as using libraries like tensor flow which can support high-performance graphics processing units (GPUs).

References

Cantwell, B. J. (1992). Exact solution of a restricted euler equation for the velocity gradient tensor. *Physics of Fluids A: Fluid Dynamics*, 4(4):782–793.

Chevillard, L., Lévêque, E., Taddia, F., Meneveau, C., Yu, H., and Rosales, C. (2011). Local and nonlocal pressure hessian effects in real and synthetic fluid turbulence. *Physics of Fluids*, 23(9):095108.

Chevillard, L. and Meneveau, C. (2007). Intermittency and universality in a lagrangian model of velocity gradients in three-dimensional turbulence. *Comptes Rendus Mécanique*, 335(4):187–193.

Chevillard, L., Meneveau, C., Biferale, L., and Toschi, F. (2008). Modeling the pressure hessian and viscous laplacian in turbulence: comparisons with direct numerical simulation and implications on velocity gradient dynamics. *Physics of Fluids*, 20(10):101504.

Eames, I. and Flor, J. B. (2011). New developments in understanding interfacial processes in turbulent flows.

Ireland, P. J., Vaithianathan, T., Sukheswalla, P. S., Ray, B., and Collins, L. R. (2013). Highly parallel particle-laden flow solver for turbulence research. *Computers & Fluids*, 76:170–177.

Johnson, P. L. and Meneveau, C. (2016). A closure for lagrangian velocity gradient evolution in turbulence using recent-deformation mapping of initially gaussian fields. *Journal of Fluid Mechanics*, 804:387–419.

Lawson, J. and Dawson, J. (2015). On velocity gradient dynamics and turbulent structure. *Journal of Fluid Mechanics*, 780:60–98.

Müller, A. C., Guido, S., et al. (2016). *Introduction to machine learning with Python: a guide for data scientists*. " O'Reilly Media, Inc.".

Wilczek, M. and Meneveau, C. (2014). Pressure hessian and viscous contributions to velocity gradient statistics based on gaussian random fields. *Journal of Fluid Mechanics*, 756:191–225.

I adhered to the honor code in the report.