CS5785: Applied Machine Learning Final Report

December 11, 2019

Abstract

The research and analysis found in this paper supports our efforts to build a large-scale image search engine to search for relevant images based on natural language queries provided by users. The goal here is to provide users with the images most similar to the query that they input . For example, if a user searches 'Man playing basketball', some images we might expect to see would be of players in the NBA or children playing basketball. However, if five of the top images show a woman eating pasta, the image search engine is most likely not effectively finding the images most similar to the query. Below we will walk through our approach and provide an analysis to support our conclusion that the optimal model for our implementation and preprocessing choices uses Kernel Ridge Regression.

1 Introduction

As noted above, the goal of this project was to train a machine learning model to support a search engine in identifying the images most similar to a natural language query provided by a user. In order to tackle this problem, we must first evaluate the inputs provided to us to better understand which inputs to use and in what fashion (discussed in sections 1.1 and 1.2). Finally, we will want to identify and implement different algorithms for our search engine and evaluate their effectiveness (discussed in section 1.3).

It is important to note that this process was iterative in nature, requiring us to test our hypotheses and revise the inputs, combinations of pre-processing, and algorithm paramaters used. Cross validating the parameters used helped us optimize and fine-tune our algorithms.

1.1 Inputs

To train our model, we used 10,000 samples, each of which has four corresponding data files for our use. The four different files are listed below for reference:

1. A 224x224 JPG image
2. A list of tags indicating objects that are present in the image
3. Feature vectors extracted using ResNet, a state-of-the-art Deep-learned CNN. The features are extracted from pool5 and fc1000 layer
4. A five-sentence description of the image

With the solution template provided to us, we take the five sentence descriptions and use word2vec to convert the descriptions to numeric feature vectors using pre-trained 300-dimensional vector representations of most words in the English language. This will serve as our x_train.

For our y_train, we use a random projection of the 1,000 ResNet features to 250 dimensions (altered from the original 100 dimensions for better performance).

Also, we will briefly mention that we attempted to make use of the list of tags by appending the words in the tag file to the corresponding array of words from the description text file. However, this did not improve performance, and so we choose to disregard the tags.

1.2 Preprocessing

One of the most important aspects of this exercise is proper preprocessing of the inputs – we found that even with a good algorithm, our results were much better when the data was properly pre-processed. In order to do so, there were several actions we performed prior to using word2vec to transform the descriptions (x_train) to numeric feature vectors. Additionally, there were other pre-processing actions we evaluated that did not improve performance and were therefore removed. The actions that did improve our score and which we used in our optimal submission include:

1. Converting all letters to lowercase
2. Removing punctuation
3. Removing all stop words (in order to do so, we used the 'Natural Language Toolkit' (nltk) library) [1]

After we performed these preprocessing steps and converted the descriptions to numeric vectors using word2vec, we made use of the sklearn.preprocessing library to import the PolynomialFeatures function to add quadratic features to the data set [2] . The goal here was to add quadratic features to better explain some of the variance in the data. Once all of these steps were completed, we then normalized the feature vectors.

The reason for converting all letters to lowercase is to ensure that words like 'Banana' and 'banana' are recognized correctly as the same word and that the capitalization here does not fundamentally change the meaning of the word. The removal of punctuation from the descriptions is to ensure that words followed by punctuation (a period, comma, etc.) are not recognized as separate words from those without punctuation (for example, 'banana', 'banana,', and 'banana.' all hold the same meaning). The goal of removing stop words is to prevent the dilution of important words in the description with words such as 'the' and 'or'. Ultimately, we wanted to make sure that we captured the most critical words and their proper counts, while also capturing the intent of query into the feature vector. Once we cleaned the description, we then used word2vec to convert the text to a numeric feature vector. At this point, we then proceeded with normalizing the feature vectors by using l2 normalization. In a 'Medium' article titled, 'Why Data Normalization is necessary for Machine Learning models,' the author states, 'Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.'[3] Given that our columns/features did not have a common scale, we wished to avoid this distortion in ranges, and so we normalized the data.

Lastly, while we did not end up using several preprocessing actions that we tested, we would like to mention them briefly and explore why they did not improve the performance of the model. Some additional preprocessing exercises that we attempted but ultimately removed for performance purposes are listed below:

1. Removing digits from the descriptions
2. Lemmatizing the descriptions

Ultimately, removing digits from descriptions was not effective because it removed key information from descriptions where the digits are an integral part of the description. Below, we have provided such an example that distinguishes between 'Usain Bolt 100m' and 'Usain Bolt m' where the 100 has been removed. In this case, the 100 in 100m is defining the event and is critical information. Clearly we can see that while Usain Bolt is still captured, most of the images are not the same.

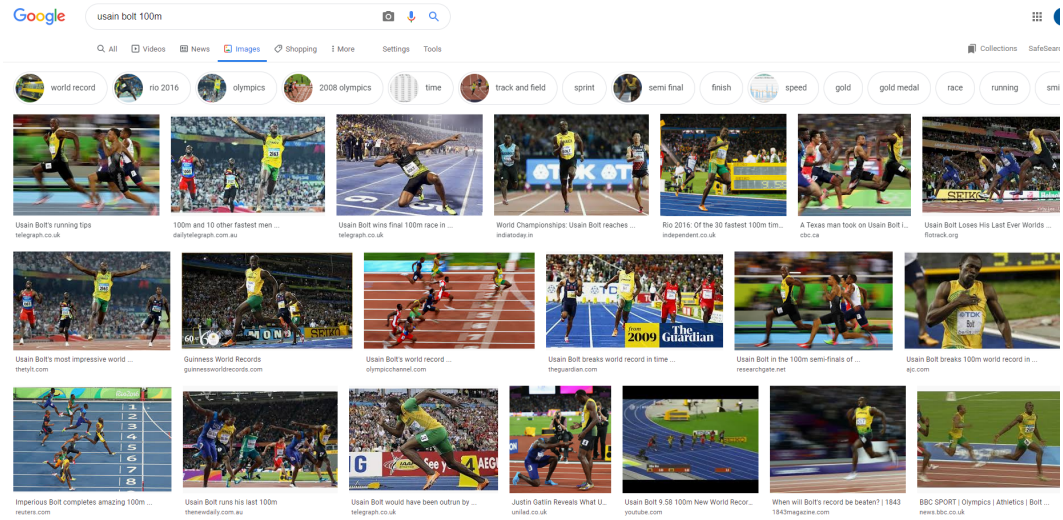


Figure 1: 'Usain Bolt 100m'

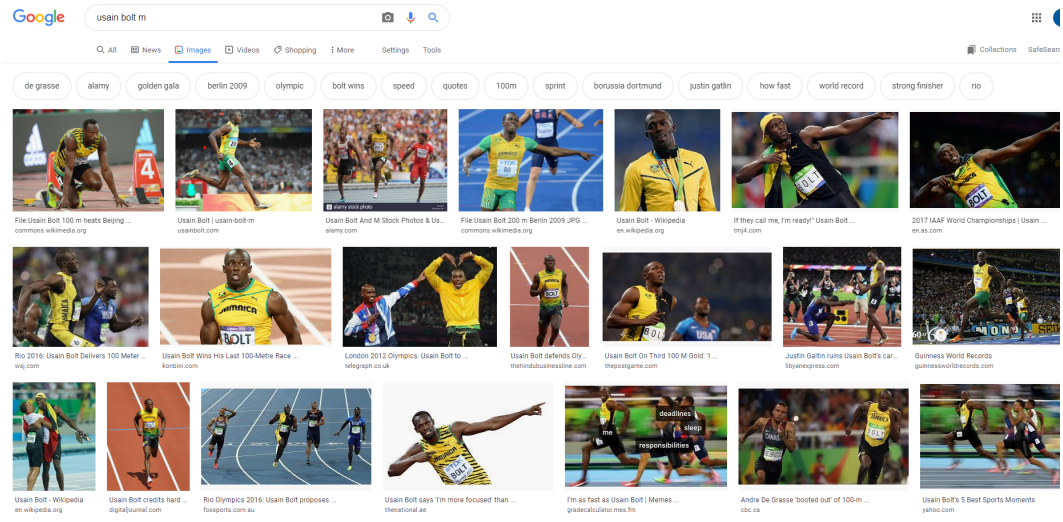


Figure 2: 'Usain Bolt m'

Regarding lemmatization, the Stanford Natural Language Processing Group defines lemmatization as 'doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.' [4] With lemmatization, our goal was to remove the ends of words so that words such as 'write' and 'writing' would be captured as equivalent words (as they convey similar meaning). However, what we found is that the tense of the verb/word is critical to capturing the desired output, especially when it relates to images. For example, we have provided two figures below that display the output of the searches 'Man run' and 'Man running'. While they share overlap, they clearly do return different results, and thus had we persisted with lemmatizing, we would have returned different images from those most similar to the natural language query provided.

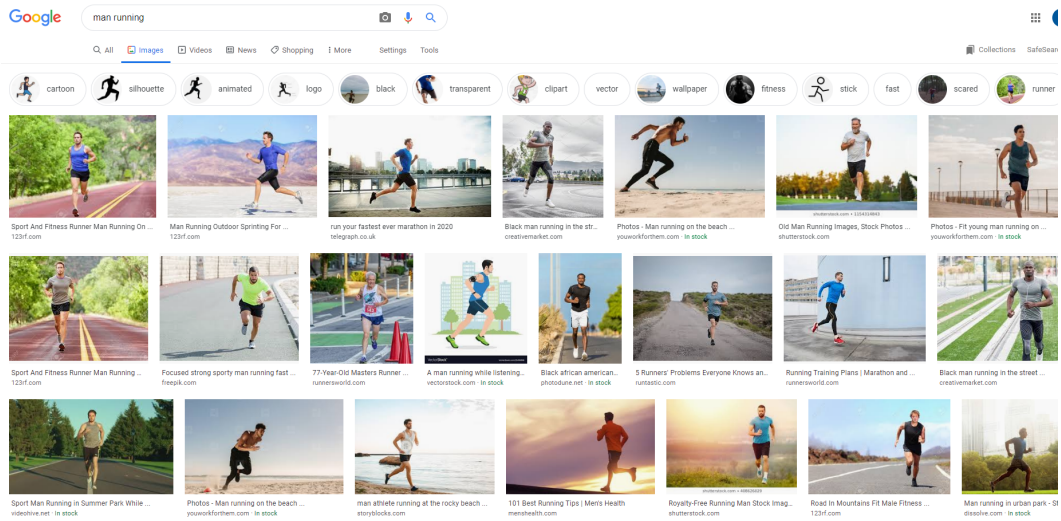


Figure 3: 'Man running'

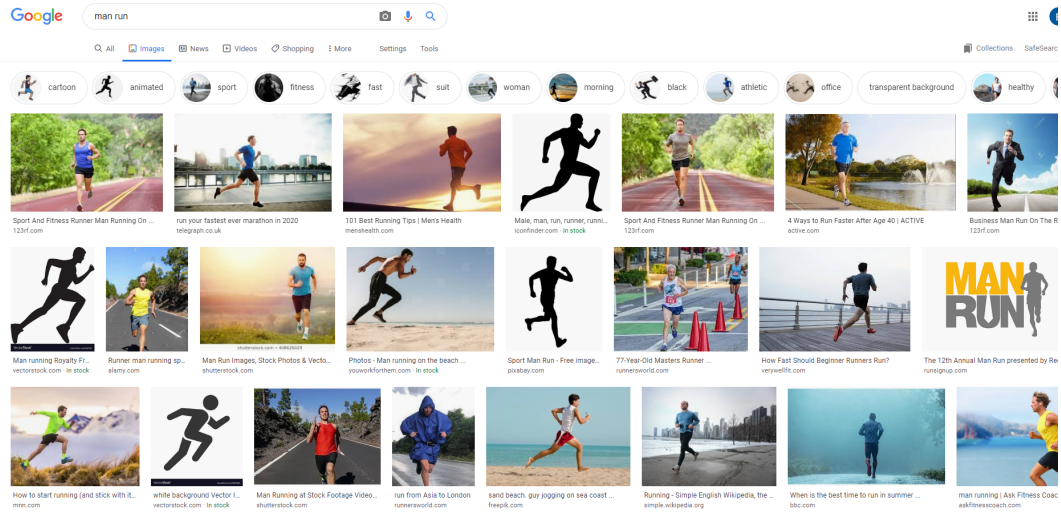


Figure 4: 'Man run'

1.3 Regression Models

Once we pre-processed the data into the desired format, we then proceeded with testing the performance of different algorithms on the newly formed training data. Below we list the algorithms that we focused on testing (we will discuss their performance and analyze them further in the experimentation and results sections):

- Lasso Regression
- Ridge Regression
- KNeighbors Regression
- Kmeans $\rightarrow$ Ridge Regression
- Kmeans $\rightarrow$ KNeighbors Regression
- Kmeans $\rightarrow$ Kernel Ridge Regression
- Random Forest Regression
- PCA $\rightarrow$ SGD (Stochastic Gradient Descent) Regression
- PCA $\rightarrow$ SVR (Support Vector Regression)
- Neural Networks
- Partial Least Squares Regression
- Kernel Ridge Regression

2 Experimentation

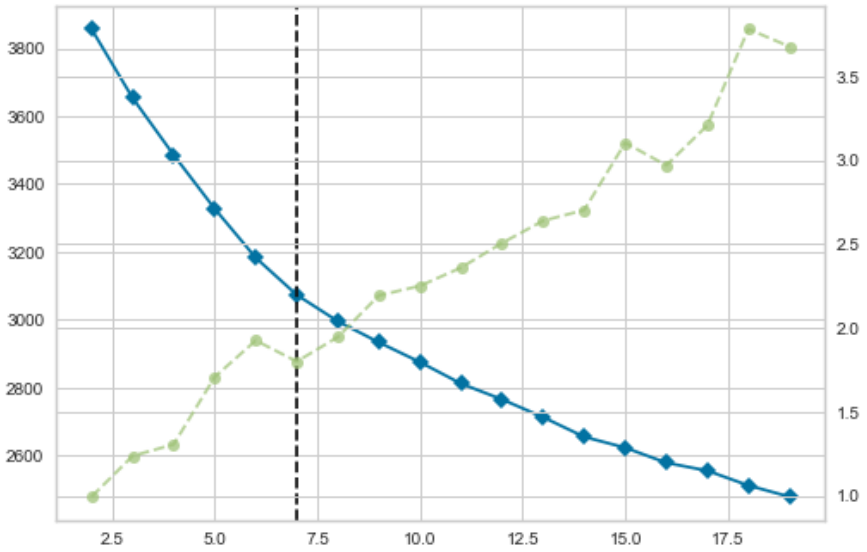
During the experimentation phase, we focused most of our efforts on two major components:

1. Testing different combinations of pre-processing for the inputs:
   - For example, we tried running different algorithms with and without lemmatization and with and without removing digits of the input descriptions. Ultimately, we saw that performance was worse when we did, so we left both out of our final runs

2. Using cross validation and fine-tuning the relevant parameters for each of the algorithms listed to achieve the optimal results. Below we will list the relevant parameters we fine-tuned for each algorithm and explain our experimentation process:
   - Ridge Regression: 'alpha' [5]
     - After cross validating, we found that our optimal alpha value was 1.12
   - KNeighbors Regressor: 'n_neighbors' for number of neighbors and 'weights' for weight of each neighbor [6]
     - We found that setting 'weights' equal to 'distance' performed better (with this implementation, neighbors are weighed by the inverse of their distance as opposed to a uniform weight system, which is the default)
     - After cross validating, we also found that our optimal 'n_neighbors' value was 3
   - Random Forest Regressor: 'n_estimators' and 'max_depth' [7]
     - Random Forest Regressor runs very slowly, which made it quite difficult to cross-validate effectively. While we were able to achieve somewhat reasonable results by fine-tuning the parameters, the results weren't better than some of our other results. Given its performance and run-time issues, we opted to focus on other algorithms instead
   - Neural Networks: 'hidden_layer_sizes', 'activation', 'solver', 'alpha', 'learning_rate' [8]
     - We experimented with different numbers of layers and neurons within each layer
     - We experimented with which activation functions to use ('identity', 'logistic', 'tanh', 'relu'). Ultimately, 'relu' worked best for us
     - We experimented with which solvers to use ('lbfgs', 'sgd', 'adam'). Ultimately, 'adam' worked best for us
     - We experimented with which learning rate to use. 'adaptive' worked best because lowering the learning rate adaptively generally leads to better results
     - Lastly, we experimented with which alpha value to use and found that an alpha of 0.1 was optimal for our model
   - Partial Least Squares Regression: 'n_components' [9]
     - After cross validating, we found that our optimal n_components value was 300. This model performed quite well, but it was not as good as our best model and took a bit longer to run
   - Kernel Ridge: 'alpha', 'kernel', 'gamma', 'degree' [10]
     - After cross validating, we found that our optimal alpha value was 0.031 and our optimal gamma value was 0.1
     - We experimented with several different types of kernels ('poly', 'linear', 'sigmoid', 'laplacian', 'rbf')
     - We also experimented with different degree values when we used the 'poly' kernel. Ultimately, the optimal degree value we found was 6
     - Before we added quadratic features, our optimal model used the 'poly' kernel. However, with quadratic features, using 'rbf' with the gamma and alpha values above proved to be our optimal model

- Lasso Regression: 'alpha' [11]
  - Cross validating for Lasso was quite slow. We tried with a few alpha values and quickly realized that it performed poorly

- Kmeans → Ridge Regression: 'n_clusters', 'alpha' [5][12][13]
  - As a clever/different approach, we hypothesized that splitting our data into clusters and then running regressions for each cluster might be more successful. So, we ran Kmeans and split our data into 7 clusters (we determined the optimal number of clusters using YellowBrick). We then built a regression model for each of those 7 clusters. For each test sample, we then found which cluster center it was closest to and made a prediction on that sample using the regression model for that specific cluster. Below is the elbow curve we used to determine that the optimal number of clusters was 7:



  - The results for this method were approximately 1 percentage point higher than running a standard ridge regression on all of the data together

- Kmeans → kNeighbors Regresion: 'n_clusters', 'n_neighbors' [6][12][13]
  - The same process as above was used. Ultimately, however, using Kmeans actually made the model perform a bit over 1 percentage point worse than when we ran the KNeighbors regression for the whole dataset

- Kmeans → KernelRidge Regresion: 'n_clusters', 'alpha', 'kernel', 'gamma', 'degree' [6][12][13]
  - The same process as above was used. We cross validated our alpha, gamma, and degree values for the 'poly' and 'rbf' kernels, as they performed best. However, using kMeans made the model perform just about the same as when we ran the KernelRidge regression for the whole dataset

- PCA $\rightarrow$ SGD (Stochastic Gradient Descent) Regression: 'n_components', 'alpha', 'max_iter' [14][15]
  - We wanted to experiment with SGD as well. However, the model seemed to only accept inputs of the shape (X,). Therefore, we needed to use PCA to reduce our data to such a dimension, and so we needed 'n_components' = 1
  - We then used SGD and tested out several different combinations of values. However, the results were very poor, most likely because we had to reduce our dimensions so much. We chose to move away from SGD because of this

- PCA $\rightarrow$ SVR (Support Vector Regression): 'n_components', 'kernel', 'degree', 'gamma', 'C' [14][16]
  - Similarly, SVR required us to use PCA in the same way. Our results were again consistently poor, so we chose to move away from SVR as well

3 Analysis & Results

Below we list the algorithms that we tested from best to worst in terms of performance. We have omitted the numeric results for our top performers to maintain anonymity in the review process. Please note that these results are specific to our implementation and under the constraints of our framework. Additionally, we must acknowledge that there are many other algorithms we did not fully test. With that in mind, the listing below should not be taken to mean that for this exercise, Neural Networks will always outperform Random Forests or that Kernel Ridge will always outperform Partial Least Squares regression.

Algorithm	**Accuracy Score**
Kernel Ridge Regression	*****
Kmeans $\rightarrow$ Kernel Ridge Regression	*****
Partial Least Squares Regression	*****
Neural Networks	*****
Kmeans $\rightarrow$ Ridge Regression	0.18365
Ridge Regression	0.17214
KNeighbors Regression	0.15382
Kmeans $\rightarrow$ KNeighbors Regresion	0.13884
Random Forest Regression	0.07356
Lasso Regression	0.00680
PCA $\rightarrow$ SVR	0.00286
PCA $\rightarrow$ SGD Regression	0.00240

As can be seen from the table above, the choice of algorithm was very influential in the overall performance of the model. Walking up the table – PCA with SGD and SVR performed poorly in large part due to the number of components we used for PCA. Lasso regression performed poorly due to two notable factors. First, Lasso was very computationally intensive on our local machines, such that we could not fully optimize it within a reasonable amount of time. In addition, Lasso works to reduce the coefficients of certain features to 0, such that it eliminates those features from the regression. We believe that with this particular data set, even though certain features may not be very important or carry much influence, eliminating them outright hurts our results. Similarly, for Random Forest Regression, we found that trying to optimize the implementation with different depths and numbers of trees in the forest was incredibly time intensive and did not provide promising results - for that reason we moved forward with some of the other algorithms listed in our table.

We will now evaluate KNeighbors Regression and Ridge Regression. Both of these algorithms ended up in the middle of the table after using cross-validation to find the optimal alpha for Ridge and the optimal number of neighbors for KNeighbors. Both of these algorithms were much less computationally intensive to run, and because of that, we were better able to optimize their parameters to achieve better results. Furthermore, Ridge outperformed Lasso in this case because of the nature of the data and the fact that it did not outright eliminate features. Additionally, because the goal is ultimately to find the pictures most similar to a given picture based on description, it would make sense that KNeighbors would work fairly well as we do want to find the nearest neighbors for the image. However, both of these algorithms plateaued in large part due to the non-linearity of the data we are working with.

Our next approach involved starting with unsupervised learning through Kmeans clustering. As was noted in the experimentation section, our hypothesis was that by clustering first, we would be able to run multiple regressions on different sections of the data based on where they clustered. Once we clustered the data, we then ran Ridge Regression, KNeighbors regression, and Kernel Ridge Regression, which had mixed performance, with the clustering improving Ridge's performance but hurting KNeighbors' and Kernel Ridge's performances. Ultimately, the increase to Ridge was not significant enough for us to continue with this approach.

Lastly, we will look at our top 3 models - Kernel Ridge, PLS, and Neural Networks. The overarching reason for the success of these three implementations is in large part due to the ability of each algorithm to effectively handle the non-linearity of the data. As we saw in the experimentation section for Kernel Ridge, when we used the 'poly' kernel, the most optimal degree value we found was 6. By using a higher polynomial degree in the regression, we were able to better fit the data and thus achieve better results. Similarly, in a 'Medium' article titled 'How neural networks learn nonlinear functions and classify linearly non-separable data?' the author states the following: 'neural network learning is a special case of kernel trick which allows them to learn nonlinear functions and classify linearly non-separable data.' [17] From here we can see that in addition to the many parameters we fine-tuned and discussed in the experimentation section, neural networks are well suited for the nonlinear data were working with. Lastly, for Partial Least Squares Regression, the ability to well predict nonlinear data resulted in strong results for this exercise. This is further supported by the article 'Study of partial least squares and ridge regression methods' written by Luis Firinguetti, Golam Kibria, and Rodrigo Araya and posted on Taylor and Francis Online (part of the Academic Publishing Division of Informa), where they state: 'From the simulation study it is evident that the RR performs better when the error variance is large and the PLS estimator achieves its best results when the model includes more variables.'[18] Because our model includes many variables, PLS performed well, as highlighted in their simulation study.

In summary, we tried many different algorithms and experimented with many different preprocessing techniques, and optimized different input parameters for each algorithm used. We found a fair deal of variability in performance between different algorithms and different input preprocessing techniques. To that end, we must note that given the nature of the assignment and that half of the test data was withheld, we also had to find the right balance in ensuring that we did not overfit our models to the training data when trying to maximize results. Ultimately, we found that Kernel Ridge performed best of all of our implementations, with PLS and Neural Network implementations also producing strong results.

4 References

[1] https://www.nltk.org/

[2] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

[3] https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029

[4] https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

[5] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

[6] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

[7] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[8] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[9] https://scikit-learn.org/stable/modules/generated/sklearn.cross_decomposition.PLSRegression.html

[10] https://scikit-learn.org/stable/modules/generated/sklearn.kernel_ridge.KernelRidge.html#sklearn.kernel_ridge.KernelRidge

[11] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

[12] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[13] https://www.scikit-yb.org/en/latest/api/cluster/elbow.html

[14] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[15] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html

[16] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

[17] https://medium.com/@vivek.yadav/how-neural-networks-learn-nonlinear-functions-and-classify-linearly-non-separable-data-22328e7e5be1

[18] https://www.tandfonline.com/doi/abs/10.1080/03610918.2016.1210168?journalCode=lssp20