

Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS5141—Intelligent Systems Laboratory

**Assignment #1— Data Cleaning, Feature Engineering and Comparative Analysis of Classification Techniques**

Prepared by: Miral Issa – 1200527

Instructor: Mohammad Jubran

Assistant: Hanan Awawdeh

Date: November 29, 2024

# Abstract

This report evaluates the impact of data preprocessing, feature selection, and model selection on the performance of machine learning classification tasks.

Data cleaning and feature engineering techniques, such as handling missing values, addressing outliers, scaling, and dimensionality reduction, were applied to prepare the dataset for analysis.

The performance of three classification models—Random Forest, Support Vector Machine (SVM), and Multilayer Perceptron (MLP)—was compared using metrics like accuracy, precision, recall, and F1 score. Results indicate that Random Forest achieved the highest accuracy (89.4%) with the shortest training time (1.67 seconds), while SVM demonstrated lower memory usage but was computationally expensive for larger datasets. MLP provided a balanced trade-off between accuracy and efficiency, achieving 86.3% accuracy with moderate training time and memory requirements.

The study concludes that data preprocessing and feature selection significantly enhance model performance, and the choice of the best model depends on task-specific requirements and resource constraints.

# Contents:

Contents

[Abstract 2](#_Toc183885203)

[Contents: 3](#_Toc183885204)

[1. Introduction 4](#_Toc183885205)

[2. Procedure and Discussion 7](#_Toc183885206)

[2.1. Part 1: Data Cleaning and Feature Engineering for the Bike Sharing Dataset 7](#_Toc183885207)

[2.1.1. Data Exploration: 7](#_Toc183885208)

[2.1.2. Data Visualization: 8](#_Toc183885209)

[2.1.3. Data Cleaning: 10](#_Toc183885210)

[2.1.4. Feature Engineering: 13](#_Toc183885211)

[2.1.5. Model Evaluation: preprocessed data vs raw data 19](#_Toc183885212)

[2.2. Part 2: Comparative Analysis of Classification Techniques 20](#_Toc183885213)

[2.2.1. Random Forest: 20](#_Toc183885214)

[2.2.2 SVM models: 22](#_Toc183885215)

[2.2.3. Multilayer Perceptron (MLP): 25](#_Toc183885216)

[2.2.4. Comparing Random Forest, SVM and MLP models: 27](#_Toc183885217)

[3. Conclusion: 29](#_Toc183885218)

# Introduction

Data cleaning and feature engineering are very important steps in data preprocessing. They aim to prepare the data to be in a suitable form for the models to use, in addition they could affect the model performance, and training time.

However, choosing the correct model to process the data is still an important step. Different models handle the data in different ways, it’s natural for some modules to perform better on some data and goal, while failing where other models succeed on another data and goal.

Most models can’t handle non-numeric features/data, data cleaning aim to convert the data into numeric data while preserving its unique information, alongside dealing with any missing values, outliers, scaling or any other issues in the data.

feature selection look into reducing the high dimensionality of the data. choosing the least amount possible of features that guaranty high quality performance of the model. Promising less training time and reducing the possibility of overfitting if done right.

All this preprocessing of data won’t mean much if used for an unsuitable model. For a classification problem there exist various models to choose from.

Random Forest: an ensemble model built on multiple decision tree classifiers. Each decision tree is trained on a sub-set of the dataset, in order to introduce randomness in the tree. Furthermore, when selecting a feature for a test node during tree construction, only a subset of the features is considered. This guarantee that not all trees will suffer from the same problems, and their mistakes will cancel out.

The final prediction in random forests is usually obtained by averaging predictions from all trees. Giving a high accurate classification.

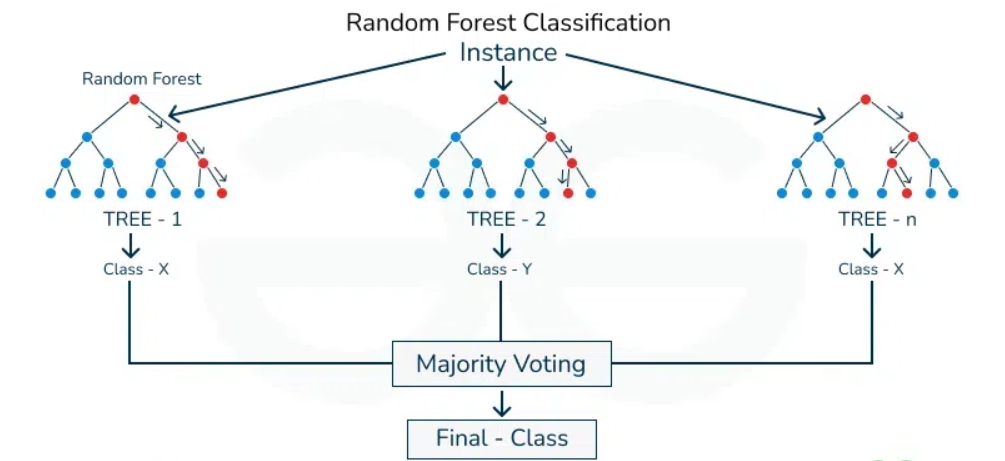


Figure ‎1.1: Random Forest model structure

SVM models: (support vector machine) SVM tries to find a separating hyper-plane between the classes with the maximum margin. Depending on the kernel trick to find this hyper-plane in the minimum costly way. However, highly entangled data can challenge this model.

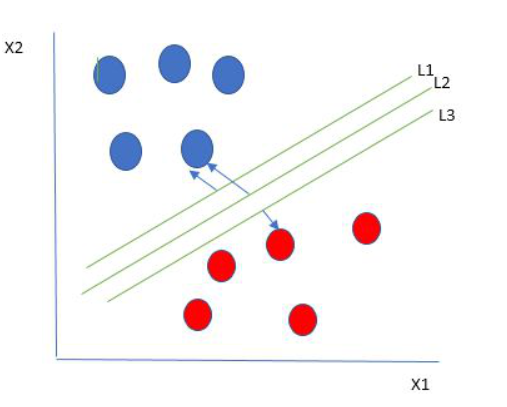


Figure ‎1.2:SVM separation margin

Multilayer Perceptron (MLP): a fully connected dense layers that transform input data from one dimension to another. Typically, it contains one input layer, one output layer, and multiple hidden layers. It has the ability to model complex relationships between inputs and outputs, making it a powerful tool for various machine learning tasks.

But if all these preprocessing steps and different models perform differently, which of them could be a better choice? How do each of them differ? What different outcomes should be expected from them?

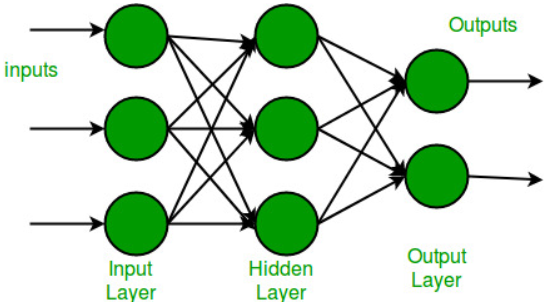


Figure ‎1.3: multi-Layer perceptron structure

# Procedure and Discussion

## Part 1: Data Cleaning and Feature Engineering for the Bike Sharing Dataset

### Data Exploration:

Using the shape property told us that the original data set had 34 features, and 7000 rows/data points. With the features being:

['Target', 'Genetic Markers', 'Autoantibodies', 'Family History', 'Environmental Factors', 'Insulin Levels', 'Age', 'BMI', 'Physical Activity', 'Dietary Habits', 'Blood Pressure', 'Cholesterol Levels', 'Waist Circumference', 'Blood Glucose Levels', 'Ethnicity', 'Socioeconomic Factors', 'Smoking Status', 'Alcohol Consumption', 'Glucose Tolerance Test', 'History of PCOS', 'Previous Gestational Diabetes', 'Pregnancy History', 'Weight Gain During Pregnancy', 'Pancreatic Health', 'Pulmonary Function', 'Cystic Fibrosis Diagnosis', 'Steroid Use History', 'Genetic Testing', 'Neurological Assessments', 'Liver Function Tests', 'Digestive Enzyme Levels', 'Urine Test', 'Birth Weight', 'Early Onset Symptoms']

Some statistics of the numeric features:

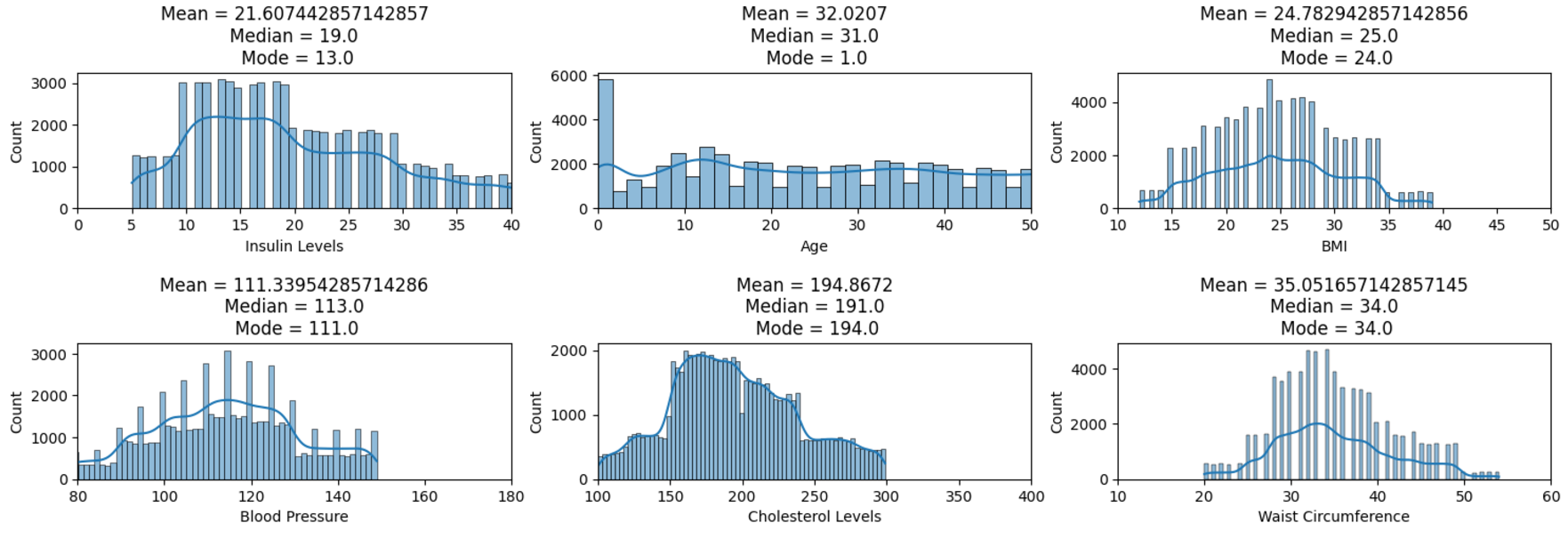


Figure ‎2.1: mean, median and mode values of numeric features

From figure 4, we could see that the numeric features are very different from each other. For example, the ‘Age’ feature has more of a normal distribution, while the ‘Cholesterol levels’ has a similar distribution to a Gaussian distribution.

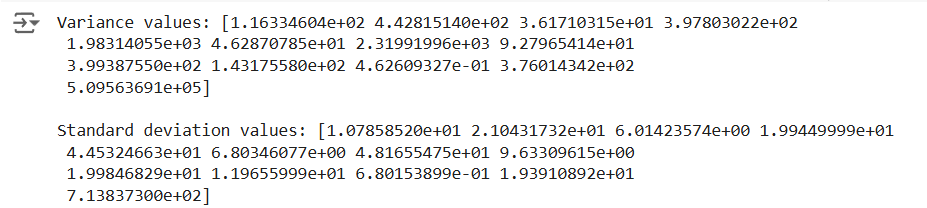


Figure ‎2.2: variance and standard deviation values of numeric values

Proving this point even more is the different variance and standard deviation of numeric values.

### Data Visualization:

To better understand the data, visualizing the data helps tremendously.

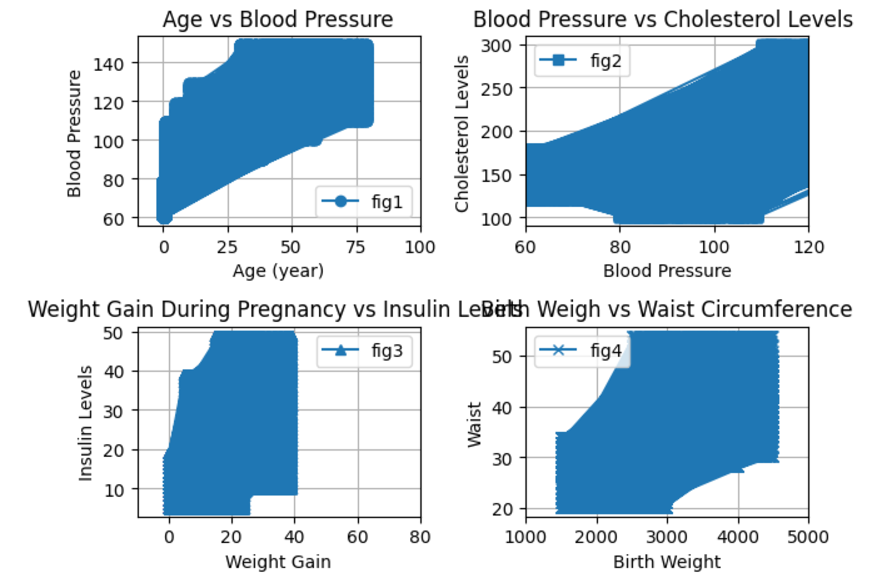


Figure ‎2.3: plots highlighting relationships between different features.

Figure 6 highlight different relationships between different features of the data set. For example, the graph shows that older ages in general suffer from higher blood pressures. Also, higher Cholesterol levels could go hand in hand with higher blood pressures. It’s very interesting that gaining more weight during pregnancy is viewed alongside higher insulin levels.

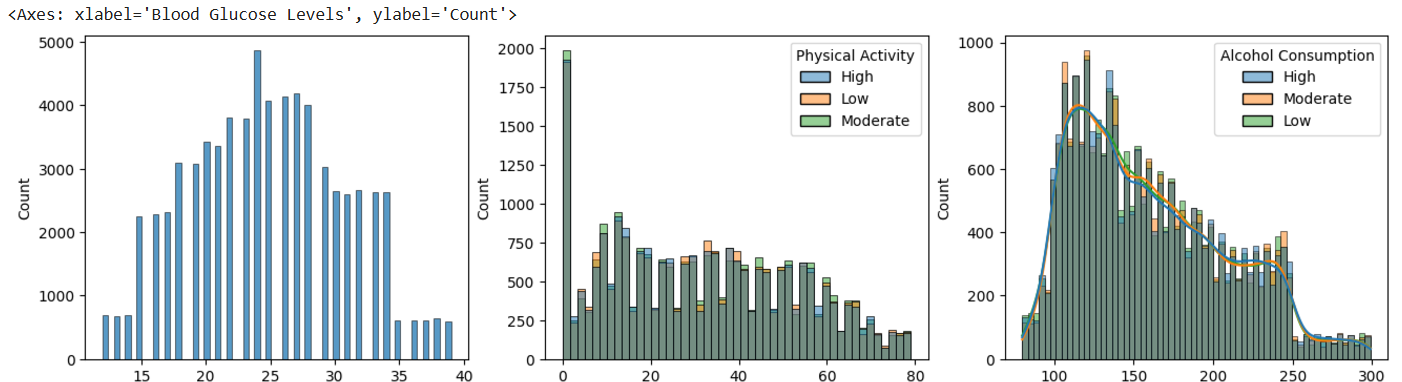


Figure ‎2.4: histograms of different features

The histograms in figure 7 help us realize that BMI have a Gaussian distribution, with BMI =~ 24 the most repeated value in the data set. Also all age groups do moderate physical activities, while a lot of data points fall into the age groups less than 1 year. Blood glucose levels distribution is skewed to the left of its mean.

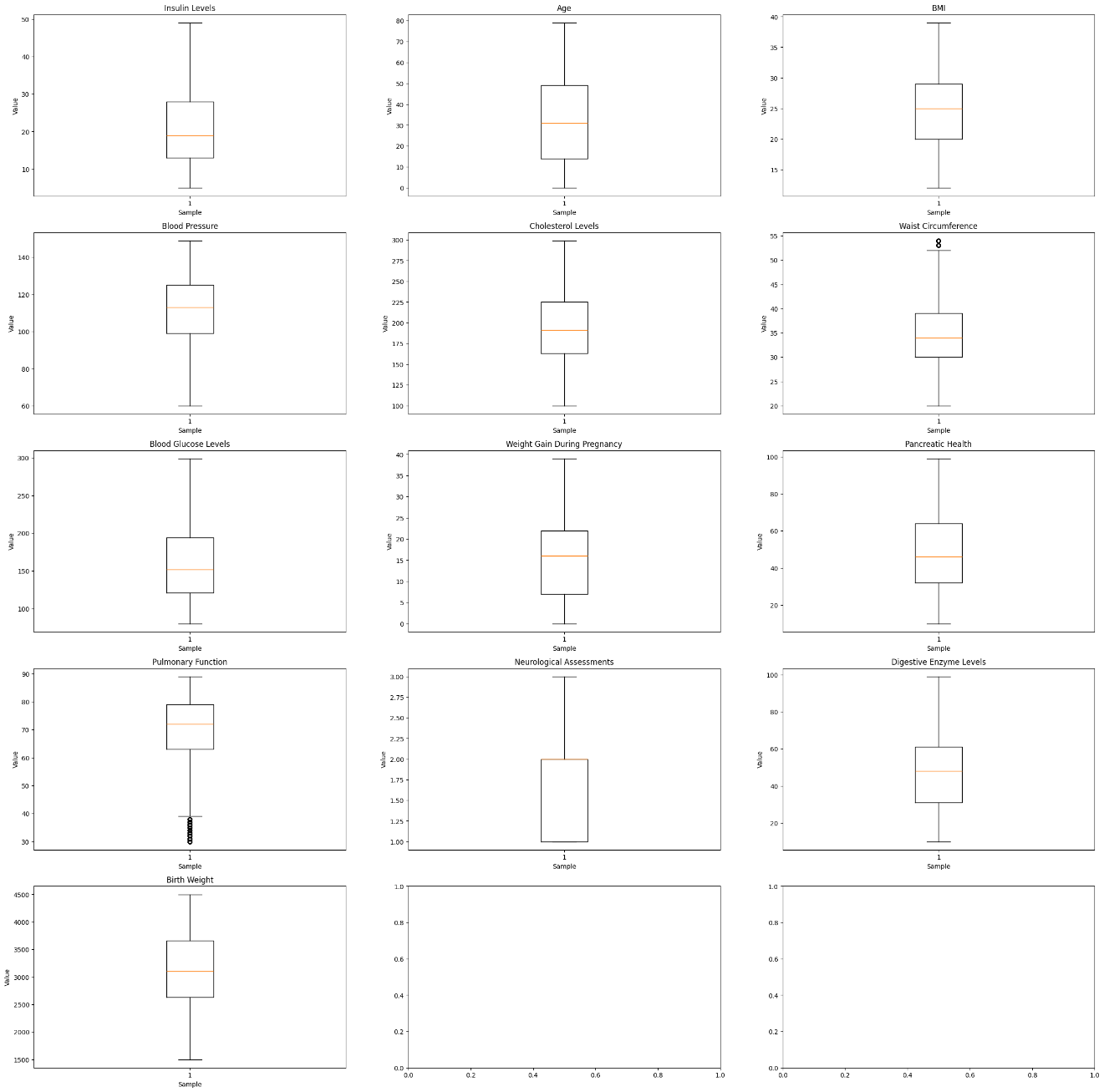


Figure ‎2.5: boxplots of numeric features

Boxplots help visualize the ranges and outliers of numeric features, with them we could easily notice that most numeric features have similar ranges, except features like: ‘Neurological Assessments’ that have very small range between 1 and 3, 'Birth Weight' that fall into a range of thousands.

### Data Cleaning:

Issues in the data must be addressed before using it to train and test different models.

#### Dealing with Missing Data:

When using df.isnull().sum()its output showed no missing values in all features. However, to be sure that no missing data were captured with some place holder values, I printed all unique values of all features, and still found that indeed the data set contained no missing values.

#### Dealing with outliers:

The boxplot was used on all numeric values to detect any outliers, and it detected outliers in two numeric features: Waist Circumference and Pulmonary Function.

In ‘Waist Circumference’:

Lower Bound = 16.5, and Upper Bound = 52.5 were the bounds found, with 522 data points with values outside these bounds, or 0.74% of all rows. As this percentage is less that 0.8%, it’s regarded as a small number of rows that can be ignored. Hence, why the outliers were deleted in this case.

In ‘Pulmonary Function.’:

Lower Bound = 39.0, and Upper Bound = 103.0 were the bounds found, with 1206 data points with values outside these bounds, or 1.735% of all rows, which is a big percentage that can’t be ignored.

Going back to the boxplot of this feature in figure 2.6, the figure shows that all outliers lay in the range between 30 and 40, while all values in the feature lay in the range between 30 and 90.

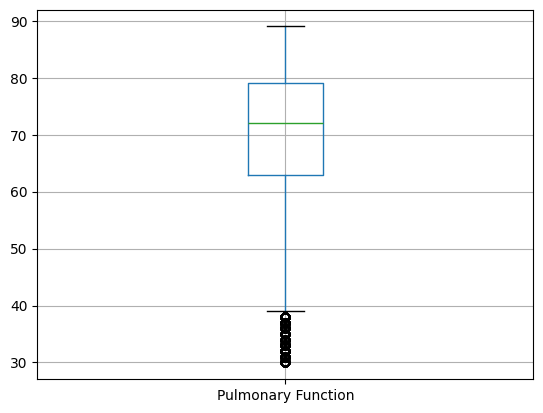


Figure ‎2.6: boxplot of 'Pulmonary Function' feature

So, if the values were transformed using a function that increase small values while keeping big values as is, all values will be inside the upper and lower bounds.

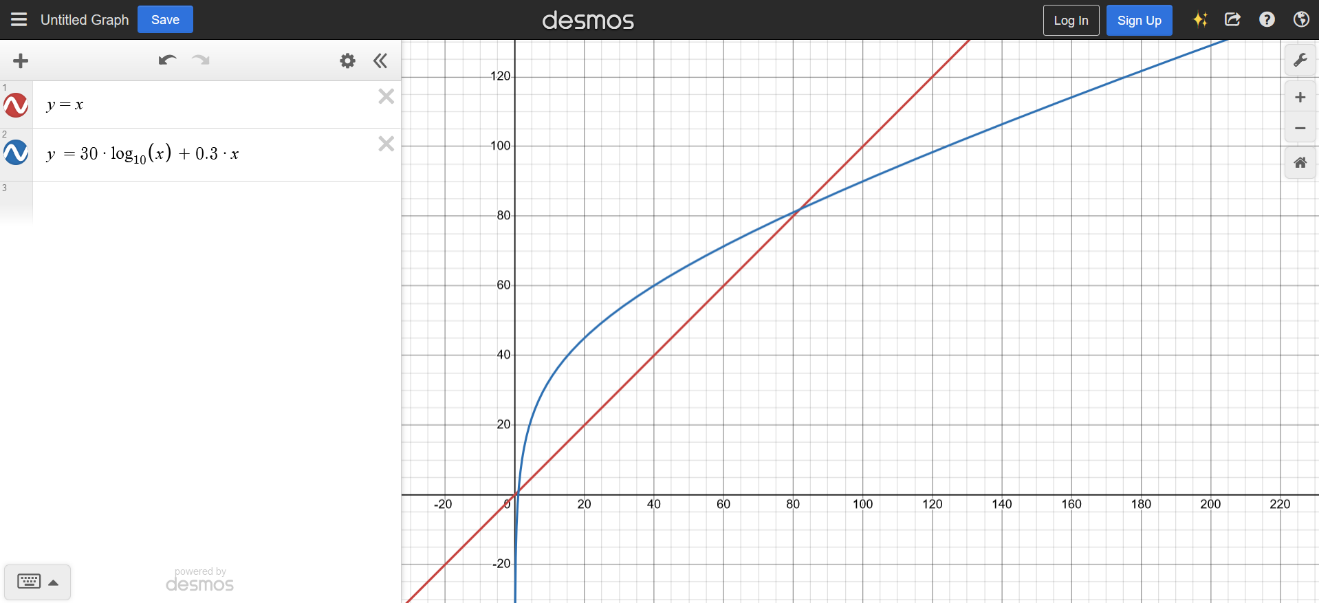


Figure ‎2.7: a mathematical transformation suitable for dealing with the Pulmonary Function feature outliers

After applying this transformation to the values of the feature no outlier outside the previously calculated outliers were preserved.

### Feature Engineering:

#### Analyze the relevance of each feature for the machine learning task:

Another good statistic to know about the numeric data, is the features variance. It gives a clearer idea of which features would really help the classification model differentiate between different classes, and which won’t due to the closeness of its values.

Table ‎2.1:numeric features variances

|  |  |
| --- | --- |
| Numeric feature | variance |
| 'Insulin Levels' | 4.4000000e+01 |
| 'Age' | 7.9000000e+01 |
| 'BMI' | 2.7000000e+01 |
| 'Blood Pressure' | 8.9000000e+01 |
| 'Cholesterol Levels' | 1.9900000e+02 |
| 'Waist Circumference' | 3.2000000e+01 |
| 'Blood Glucose Levels' | 2.1900000e+02 |
| 'Weight Gain During Pregnancy' | 3.9000000e+01 |
| 'Pancreatic Health' | 8.9000000e+01 |
| 'Pulmonary Function' | 2.8580000e+01 |
| 'Neurological Assessments' | 4.6053485e-01 |
| 'Digestive Enzyme Levels' | 8.9000000e+01 |
| 'Birth Weight' | 2.9990000e+03 |

From table 2.1 we could notice that for example, ['BMI', 'Cholesterol Levels', 'Blood Glucose Levels', 'Pulmonary Function', 'Neurological Assessments', 'Birth Weight'] are all numeric features with variance less than 3e+1 threshold. So, if a threshold of this value were applied to the data, these features would be removed from the data set. How would that affect the model would be answered in later parts of this report.

#### Encode categorical variables:

Most features in the data set are non-numeric features that aren’t suitable for the classification models used, and all of them are categorical features. In order to convert them into a data type that the models can handle, one most encode these features.

1. binary categorical features:

can be simply encoded with 0 and 1, features in the list:

['Genetic Markers', 'Autoantibodies', 'Family History', 'Environmental Factors', 'Dietary Habits', 'Ethnicity', 'Smoking Status', 'Glucose Tolerance Test', 'History of PCOS', 'Previous Gestational Diabetes', 'Pregnancy History', 'Cystic Fibrosis Diagnosis', 'Steroid Use History', 'Genetic Testing', 'Liver Function Tests', 'Early Onset Symptoms']

Were encoded in this manner.

1. Categorical features with ordered categories:

Physical Activity, Socioeconomic Factors, and Alcohol Consumption all have categories similar to low, medium and high, so they were encoded into 0, 1 and 2 respectively.

1. Multi categorical features:

These are features that contained too many categories that aren’t necessary ordered, so encoding them with numbers may affect the models negatively.

Such as ‘Urine Test’. This feature was encoded using one-hot-encoder, as teach category point to something that exist while others don’t in a urine sample, and the number of categories wasn’t too big, using one-hot-encoder made sense.

Only feature left now is the ‘Target’ feature, which contained the classes or labels of the data set, that the classifier model needs to learn do give to testing data points. This feature could be encoded with whole numbers because it wasn’t used in training the model in a sense that would mislead the model to think that the classes are ordered. Encoding the ‘Target’ feature in this manner turned the classification problem to be similar to the hand written numbers classification model, studied in one of the lab's experiments.

Now the data set no longer contains non-numeric features.

#### Rescaling:

Some models are sensitive to the scaling and ranges of the training data features. A feature with a very large scale would skew some models, while features with a very small scale may get ignored regardless of how much important to the model’s task it is.

The boxplots were very helpful in visualizing the numeric features ranges, which helped comparing them to each other, and singling out the odd ones. like the Birth Weight and Neurological Assessments features.

Both of these features were rescaled using the minmax scaler from the sklearn.preprocessing library, to be in a new range from 0 to 100.

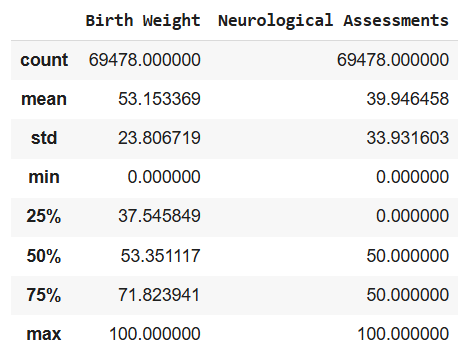


Figure ‎2.8: Birth Weight and Neurological Assessments features statistic after rescaling

#### dimensionality reduction:

dealing with 36 features after encoding is too much, and could have bad effects on the classification models, especially on its training time/computational efficiency.

What different ways there’s to remove unimportant features and keep helpful ones? There’s more than one way to do that, some of them are:

1. VarianceThreshold:

As discussed before, the variance of each feature could be a good indicator of how much this feature would help the classification model differentiate between different classes.

However, for a data set with too many categorical features (binary or multilabel) that have values very similar to each other, this method could mislabel them as unhelpful. Removing them after applying the threshold.

An experiment I did on a Random Forest classifier, outputted these differences between using all features, and using features with threshold larger than 3e+1:

Number of original features: 36

Number of features after variance threshold filtering: 13

Accuracy of Original features (testing accuracy): 0.8977403569372481

Total computational time for training model on original features: 15.7097 seconds

Accuracy after variance threshold filtering (testing accuracy): 0.8986758779504893

Total computational time for training model on chosen features: 12.5897 seconds

we could see that from 36 feature only 13 remained after the threshold. This reduction reduced the training time from 15.7097 seconds to 12.5897 seconds, yet still improved the testing accuracy from 0.8977 to 0.8986. this may be a product of reduced overfitting between the two models.

Table ‎2.2: dimensionality reduction using variance threshold improvement

|  |  |  |
| --- | --- | --- |
| Field of improvement | All features | After variance threshold |
| Number of features | 36 | 13 |
| Training time | 15.7097 seconds | 12.5897 seconds |
| Test accuracy | 0.8977403569372481 | 0.8986758779504893 |

1. Information Gain:

Another filter method for reducing data dimensionality, but unlike variance threshold, it doesn’t filter the features according to their variance value, rather according to the information gain each feature provide. This is a great way to filter features for a Random Forest model, as each tree in this model at each node depend on the information gain metric to decide which feature to use to split the data.

This filtering method remove low information gain features itself before even giving the data for the model to train on, reducing its training time without harming it.

These are the outputs of the same experiment done on information gain filter instead with k= 13 (choose the highest 13 information gain features):

Number of original features: 36

Number of features after information gain filtering: 13

Accuracy of Original features (testing accuracy): 0.8977403569372481

Total computational time for training model on original features: 13.0165 seconds

Accuracy after information gain filtering (testing accuracy): 0.8986758779504893

Total computational time for training model on chosen features: 12.5897 seconds

The filter improved the model by reducing its training time, and improving its test accuracy, by reducing the number of features again to 13, but this time according to their information gain, not variance.

Table ‎2.3: dimensionality reduction using information gain threshold improvement

|  |  |  |
| --- | --- | --- |
| Field of improvement | All features | Information gain threshold |
| Number of features | 35 | 13 |
| Training time | 13.0165 seconds | 12.5897 seconds |
| Test accuracy | 0.8977403569372481 | 0.8986758779504893 |

Another experiment was done, this time with k=10 (choose top 10 features with highest information gain), and these were the results:

Number of original features: 36

Number of features after information gain filtering: 10

Accuracy of Original features (testing accuracy): 0.8977403569372481

Total computational time for training model on original features: 11.6494 seconds

Accuracy after information gain filtering (testing accuracy): 0.8849309153713298

Total computational time for training model on chosen features: 12.5897 seconds

Table ‎2.4: dimensionality reduction using information gain with k=10

|  |  |  |
| --- | --- | --- |
| Field of improvement | All features | Information gain threshold |
| Number of features | 36 | 10 |
| Training time | 11.6494 seconds | 12.5897 seconds |
| Test accuracy | 0.8977403569372481 | 0.8849309153713298 |

This time the model accuracy decreased rather than improved, as with a too few features the model ran into an underfitting problem.

1. Feature Selection with Forward Selection:

This method works by looping over all features, train the model with one feature added at a time, then add the feature that improved the model the most to the list of best features. It takes too much time as with each loop it trains new models again and again, and keep comparing the results. But it guaranties that the best features for the classification model get chosen, and gives the improved metric that led to the addition of this feature.

These were the features chosen by this method:

Feature 'Age' added, Accuracy: 0.3460

Feature 'Blood Glucose Levels' added, Accuracy: 0.5499

Feature 'BMI' added, Accuracy: 0.6425

Feature 'Weight Gain During Pregnancy' added, Accuracy: 0.7222

Feature 'Insulin Levels' added, Accuracy: 0.7809

Feature 'Waist Circumference' added, Accuracy: 0.8223

Feature 'Digestive Enzyme Levels' added, Accuracy: 0.8520

Feature 'Cholesterol Levels' added, Accuracy: 0.8679

Feature 'Blood Pressure' added, Accuracy: 0.8777

Feature 'Pancreatic Health' added, Accuracy: 0.8854

Feature 'Pulmonary Function' added, Accuracy: 0.8931

Feature 'Birth Weight' added, Accuracy: 0.8986

Feature 'Steroid Use History' added, Accuracy: 0.9011

Feature 'Ketones Present' added, Accuracy: 0.9008

Feature 'History of PCOS' added, Accuracy: 0.9008

Feature 'Ethnicity' added, Accuracy: 0.9003

Feature 'Dietary Habits' added, Accuracy: 0.9006

Feature 'Neurological Assessments' added, Accuracy: 0.9008

Feature 'Protein Present' added, Accuracy: 0.9011

Feature 'Early Onset Symptoms' added, Accuracy: 0.9012

Feature 'Environmental Factors' added, Accuracy: 0.9012

after the accuracy reached 90%, it more or less stays the same, with some features decreasing the accuracy instead of increasing it, so I choose the best features as the features added until the accuracy reached 90% and no more.

These were the outputs of a random forest model trained on these features:

Number of chosen features: 13

Accuracy of model (testing accuracy): 0.9010506620610248

Total computational time for training the random forest model on the chosen features: 13.1889 seconds

Notice that the accuracy of the model indeed reached 90% on the testing data.

### Model Evaluation: preprocessed data vs raw data

I trained two Random Forest classification models, one on the preprocessed data with the features chosen by forward selection, and the other with the raw data only with non-numeric features encoded so the model can handel them

Table ‎2.5: Random Forest model on raw data vs preprocessed data.

|  |  |  |
| --- | --- | --- |
| Field of comparison | Raw data | Preprocessed data |
| Number of features | 36 | 13 |
| Training time | 21.2459 seconds | 13.1889 seconds |
| Accuracy | 0.8989285714285714 | 0.9010506620610248 |
| Precision | 0.9032928592224001 | 0.9049685679362848 |
| Recall | 0.8993267971221521 | 0.900289902583114 |
| F1 Score | 0.8978114702629749 | 0.8992582775564516 |

The preprocessed data improved the model on all metrics, and in the same time increased its computational efficiency. From the experiment we conclude that the preprocessed data is better than the raw data for the Random Forest classifier.

## Part 2: Comparative Analysis of Classification Techniques

### Random Forest:

#### Experiments:

A random forest model with these parameters:

clf = RandomForestClassifier(n\_estimators=2,

criterion="entropy",

max\_features="sqrt",

max\_samples=0.8,

random\_state=0)

it took a training time of 0.2660 seconds to output these test metrics:

Accuracy: 0.8296632124352331

Precision: 0.8339373421091826

Recall: 0.8292925346998907

F1 Score: 0.826970491511428

Increasing the number of estimators to 15, didn’t increase the training time, as it now equals to 1.6511 seconds, but the test metrics now equal to:

Accuracy: 0.8947179044329303

Precision: 0.8960245492691166

Recall: 0.8936846929492804

F1 Score: 0.8927279737494369

Which is a slightly improved performance, but the number of estimators isn’t the only parameter that influence the model performance.

Changing the criterion to use 'gini' instead of ‘entropy’ decreased the training time to be equal to 3.2448 seconds. However, changing the max\_features parameter to 10 increased the training time to 3.8561 seconds. With test metrics equal now to:

Accuracy: 0.8888888888888888

Precision: 0.8896573390504028

Recall: 0.887860525727269

F1 Score: 0.8870413266495157

Random Forest model have a lot of hyperparameters that customize its performance to be suitable to the problem at hand. A bigger number of estimators can handle more complex data sets, while smaller number is more computationally efficient.

Both criterion modes perform the same with ‘gini’ being slightly faster, and max features, which is the number of features each tree consider when splitting at each node influence the training time and model complexity too.

Table ‎2.6: a comparison between different between random forest models with different parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field of comparison | n\_estimators=2, criterion="entropy", max\_features="sqrt", | n\_estimators=15, criterion="entropy", max\_features="sqrt", | n\_estimators=15, criterion="gini", max\_features="sqrt", | n\_estimators=15, criterion="gini", max\_features=10, |
| Training time | 0.2660 seconds | 1.6511 seconds | 1.6683 seconds | 3.8561 seconds |
| Peak memory usage | 8.3382 MB | 14.4677 MB | 60.2249 MB | 14.7679 MB |
| Accuracy | 0.8296632124352331 | 0.8947179044329303 | 0.8939263097294186 | 0.8888888888888888 |
| Precision | 0.8339373421091826 | 0.8960245492691166 | 0.8946991135999783 | 0.8896573390504028 |
| Recall | 0.8292925346998907 | 0.8936846929492804 | 0.8929804516419151 | 0.887860525727269 |
| F1 Score | 0.826970491511428 | 0.8927279737494369 | 0.8920293146866529 | 0.8870413266495157 |

To see how sensitive is the model to data preprocessing, two models were trained, one on all features, and the other on features chosen by forward selection.

The comparison on different metrics is made with table 2.7:

Table ‎2.7: random forest trained on all features vs best features

|  |  |  |
| --- | --- | --- |
| Fields of comparison | All features | Best features |
| Training time | 0.2574 seconds | 0.2660 seconds |
| Peak memory usage | 23.4074 MB | 8.3382 MB |
| Accuracy | 0.7571963154864709 | 0.8296632124352331 |
| Precision | 0.7704228596082289 | 0.8339373421091826 |
| Recall | 0.7573282331794685 | 0.8292925346998907 |
| F1 Score | 0.7516356874256632 | 0.826970491511428 |

It’s easy to see that the model trained on best features (a subset of features) perform better than the one trained on all features, as the first one generalizes better and need less resources.

#### Strengths and weaknesses of random forest models:

Strengths:

1. Customization: as seen in previous experiments, random forest models could be customized easily using hyper parameters, like: number of estimators, criterion, max features and others.
2. Robustness to Noise: as random forest combines the forecasts of several decision trees; it is resilient to noisy data. Because noisy data points are unlikely to alter the forecasts of every tree in the forest, they have a lower chance of affecting the overall performance of the model.
3. Non-Parametric Nature: it does not assume anything about the distribution of the data at the root level or the correlation between the target variable and characteristics; it can be applied to a variety of datasets and problem areas

Weaknesses:

1. Computational Complexity: Using a large number of trees in the forest or training a Random Forest model on a large dataset can be computationally expensive. Since each tree is trained separately, it takes a lot of computing power to aggregate its predictions.
2. Memory Usage: Random Forest models have a tendency to use a lot of memory, particularly when working with big datasets or deeply rooted trees. The training data, feature splits, and leaf node predictions must all be stored in each decision tree in the forest.
3. Overfitting: Random Forest can suffer from overfitting when the model captures noise in the training data, leading to poor generalization on new data. as seen with the model trained on all features.

### 2.2.2 SVM models:

#### 2.2.2.1. experiments:

Experimenting on a model with linear model and parameters equal to:

clf = svm.SVC(kernel="linear", C=10, random\_state=42)

it took a long training time compared to all random forest models equal to 68.2861 seconds. Not much hyper parameters could be changed with an SVM model, only the type of kernel, and the C parameter, which indicate how much tolerant the model could be with data points inside the margins, or on the wrong side of the decision line.

Table ‎2.8:a comparison between different between SVM models with different parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Field of comparison | kernel="linear", C=10 | kernel="linear", C=50 | kernel="rbf", C=10 |
| Training time | 68.2861 seconds | 201.0589 seconds | 28.9298 seconds |
| Peak memory usage | 23.5123 MB | 31.0460 MB | 31.1032 MB |
| Accuracy | 0.7586355785837651 | 0.7594271732872769 | 0.8376511226252159 |
| Precision | 0.7555070537990212 | 0.7563029012480279 | 0.8376421874558008 |
| Recall | 0.7557458403727829 | 0.75657510372213 | 0.8361678018193724 |
| F1 Score | 0.7553276812457526 | 0.7561299736577329 | 0.8352283150823211 |

As seen in table 2.8, the SVM model is very sensitive to its parameters, is increasing C increased its training time and memory usage a lot, and changing the kernel to a non-linear gaussian one increased all it tests metric signifyingly.

Two more models were trained this time on increased number of features. Results are shown in table 2.9:

Table ‎2.9: : SVM trained on best features vs one more feature

|  |  |  |  |
| --- | --- | --- | --- |
| Field of comparison | Best features, with rbf kernel | One more feature with linear kernel | One more feature with rbf kernel |
| Training time | 28.9298 seconds | 3282.0241 seconds | 30.8022 seconds |
| Peak memory usage | 31.1032 MB | - | 48.0381 MB |
| Accuracy | 0.8376511226252159 | 0.7592112838226828 | 0.7848301669545192 |
| Precision | 0.8376421874558008 | 0.7560363112498988 | 0.7833514557542743 |
| Recall | 0.8361678018193724 | 0.7563845983078725 | 0.7820302120960783 |
| F1 Score | 0.8352283150823211 | 0.7559464944140447 | 0.7815057182443031 |

As seen in the table, the SVM model is also very sensitive to the data it trains on, as adding only one more feature into the training data, decreased the computational efficiency a great deal. It also decreased all test metrics, proving that that one more feature led the model into overfitting.

#### 2.2.2.2. Strengths and weaknesses of random forest models

Strengths:

1. Kernel trick: The kernel trick is real strength of SVM. With an appropriate kernel function, we can solve any complex problem. The kernel trick maps the data into a higher-dimensional space where the data becomes linearly separable.
2. Support vector machine works comparably well when there is an understandable margin of dissociation between classes.
3. Handling small datasets: SVMs can perform well with small datasets, as they only require a small number of support vectors to define the boundary.
4. Versatility: SVMs can be used for both classification and regression tasks, and it can be applied to a wide range of applications such as natural language processing, computer vision, and bioinformatics.

Weaknesses:

1. SVM algorithm is not acceptable for large data sets.
2. It does not execute very well when the data set has more sound i.e. target classes are overlapping.
3. As the support vector classifier works by placing data points, above and below the classifying hyperplane there is no probabilistic clarification for the classification.
4. Computationally expensive: SVMs can be computationally expensive for large datasets, as the algorithm requires solving a quadratic optimization problem. As seen with previous experiments.
5. Choice of kernel: The choice of kernel can greatly affect the performance of an SVM, and it can be difficult to determine the best kernel for a given dataset.

### Multilayer Perceptron (MLP):

#### Experiments:

MLP is another highly customizable machine learning model, with hyper parameters like: number of hidden layers, number of neurons in each hidden layer, activation function, learning rate and max iterations. Tuning these parameters could be challenging, but they guarantee that MLP could be suitable for large number of problems.

Aiming to maximize the model accuracy and precision, mutable experiments were done, summarized with their outputs in tables 2.10 and 2.11:

Table ‎2.10: : a comparison between different between SVM models with different parameters part 1

|  |  |  |  |
| --- | --- | --- | --- |
| Field of comparison | Hidden layer sizes=(10,), max iter=100 | Hidden layer sizes=(10,), max iter=100, activation='relu' | Hidden layer sizes=(10,10,10), max iter=100, activation='relu' |
| Training time | 27.9520 seconds | 29.7002 seconds | 44.8687 seconds |
| Peak memory usage | 48.0381 MB | 48.0381 MB | 48.0381 MB |
| Accuracy | 0.782671272308578 | 0.7684225676453655 | 0.7852619458837076 |
| Precision | 0.781990715442229 | 0.7658882190016498 | 0.7850972060857244 |
| Recall | 0.779914764154137 | 0.7648063512677405 | 0.7828336340687969 |
| F1 Score | 0.7790387252161045 | 0.7624102948299463 | 0.7829079001124213 |

Table ‎2.11: a comparison between different between SVM models with different parameters part 2

|  |  |  |
| --- | --- | --- |
| Field of comparison | Hidden layer sizes=(10,10,10), max iter=100, activation='relu', learning rate init=0.01 | Hidden layer sizes=(30,20,15), max iter=1000, activation='relu', learning rate init=0.01 |
| Training time | 45.4991 seconds | 50.6533 seconds |
| Peak memory usage | 48.0381 MB | 58.5887 MB |
| Accuracy | 0.7935377086931491 | 0.8629821531375935 |
| Precision | 0.7952217213478 | 0.8661080977367981 |
| Recall | 0.7908202231912219 | 0.8620534791329502 |
| F1 Score | 0.7885193272278225 | 0.8608803132223888 |

With each changed hyper parameter the model performance and efficiency were changing but not by much, proving that the model is highly customizable and robust.

Another model was trained this time on all features available in the training data instead of only the ones chosen in the forward selection. The comparison between it and previous models are made in table 2.12:

Table ‎2.12: MLP model trained on all features vs best features

|  |  |  |
| --- | --- | --- |
| Field of comparison | All features | Best features |
| Training time | 95.9298 seconds | 50.6533 seconds |
| Peak memory usage | 60.2249 MB | 58.5887 MB |
| Accuracy | 0.7818796776050662 | 0.8629821531375935 |
| Precision | 0.7848458914934446 | 0.8661080977367981 |
| Recall | 0.7793277220256652 | 0.8620534791329502 |
| F1 Score | 0.7781913294803426 | 0.8608803132223888 |

Increasing the number of features led the model into overfitting, as its test metrics all decreased, with addition of increasing its training time and memory usage.

#### Strengths and weaknesses of random forest models

Strengths:

1. Versatility– MLPs are flexible in that they may be employed with several kinds of input data, such as continuous or categorical variables.
2. Generalization– Multilayered perceptions may generalize effectively to new, previously unknown data when properly trained, making them suitable for real-world applications.
3. Highly customizable: with too many hyper parameters, MLP can be customized to work with linear and non-linear data, can be scaled up by adding additional hidden layers or nodes. And in general work on various types of problems and tasks.

Weaknesses:

1. Overfitting– If the model is too complicated or the training data is too restricted, MLPs may easily overfit the training data. as seen with the model trained using all features available in the training data set.
2. Optimization of MLP hyperparameters– Including the number of nodes and layers, the activation function, and the learning rate, is necessary for peak performance. Tuning all these parameters can be challenging.

### Comparing Random Forest, SVM and MLP models:

After experimenting with different models of these three types, using different parameters and training data sets, it’s time to compare them together.

A comparison of the best of each model is made in table:

|  |  |  |  |
| --- | --- | --- | --- |
| Field of comparison | Random Forest | SVM | MLP |
| Training time | 1.6683 seconds | 28.9298 seconds | 50.6533 seconds |
| Peak memory usage | 60.2249 MB | 31.1032 MB | 58.5887 MB |
| Accuracy | 0.8939263097294186 | 0.8376511226252159 | 0.8629821531375935 |
| Precision | 0.8946991135999783 | 0.8376421874558008 | 0.8661080977367981 |
| Recall | 0.8929804516419151 | 0.8361678018193724 | 0.8620534791329502 |
| F1 Score | 0.8920293146866529 | 0.8352283150823211 | 0.8608803132223888 |

1. In terms of computational efficiency:

The fastest training time was the training time of the Random Forest model, even with 15 trees to train it only took almost 2 seconds to finish training.

On the other hand, the model that used the least amount of memory was the SVM model, with peak memory usage during training equal to 31.1032 MB, it needed almost half of what the other models needed to complete its training.

This rules out the MLP model as the least computationally efficient as it needed more training time that the Random Forest one, and more memory than the SVM model.

1. In term of test metrics:

Although not by much, the Random Forest model had the largest test metrics ( Accuracy, Precision, Recall, F1 Score) of them all in this experiment. This may be due to its ensemble structure that rule out each individual tree mistakes, improving the overall performance of the forest.

But that doesn’t mean that the Random Forest is the ‘best’ for all cases, with a problem that need minimum memory usage SVM may be better to choose. And with a problem that need to be more customizable, and that need a more complex and deep structure (like dealing with images and other data) an MLP model may be a better choice.

In this experiment however, the model that provides the best balance between prediction accuracy and computational efficiency for this dataset was the MLP, as it outputted a very good accuracy, while needing a moderate memory usage, and training time.

# Conclusion:

This report aimed to analyze the impact of data preprocessing, and feature engineering on the performance of classification models. It also compared the performance of Random Forest, SVM, and Multilayer Perceptron (MLP) models.

The results demonstrated that data preprocessing, including handling missing values, addressing outliers, encoding categorical variables, and scaling, significantly improved model performance and computational efficiency. Feature selection methods, such as Variance Threshold, Information Gain, and Forward Selection, reduced dimensionality while maintaining or improving accuracy and reducing overfitting. Also reducing computational time generously.

Among the models, Random Forest exhibited the highest accuracy and precision due to its ensemble structure that mitigates individual tree errors. SVM showed resilience in memory efficiency but struggled with computational time, especially with larger datasets or complex kernels. MLP provided a balance between accuracy and customizability but required careful hyperparameter tuning to avoid overfitting.

The theoretical understanding of these models aligned with the experimental results. Random Forest's ensemble learning, SVM's kernel trick, and MLP's ability to handle non-linear relationships were validated. However, some contradictions arose, such as the SVM model's sensitivity to additional features, which increased computational demands disproportionately. This may be due to the big size of the data that challenged the SVM model that typically handle small data sets better.

In the end, no single model universally excels, and the choice of the best model depends on the dataset characteristics and the task's specific requirements.