Diabetes Classification

Brandon Le

AOS C111: Introduction to Machine Learning for Physical Sciences

December 2025

Abstract

I applied machine learning techniques to investigate how lifestyle, medical conditions, and demographic factors relate to the diagnosis of diabetes in U.S. adults. Using the balanced CDC Diabetes Health Indicators dataset (56,553 survey responses derived from the BRFSS 2015 survey), I framed diabetes status as a binary classification problem, predicting whether an individual has diabetes or prediabetes (1) versus no diabetes (0). I trained six models: logistic regression, decision tree, random forest, linear support vector machine (`LinearSVC`), k-nearest neighbors, and a multilayer perceptron (MLP) neural network. All models achieved similar performance, with test accuracies between roughly 73% and 75% and ROC curves well above chance. The best overall model was the MLP classifier, which reached 75.32% accuracy with a single hidden layer of 2,500 neurons and a validation fraction of 0.2, only scarcely outperforming logistic regression (74.58%) and linear SVM (74.61%). These results suggest that, for this dataset, relatively simple linear models can approach the performance of more complex neural networks, and that future gains will likely depend more on higher-quality clinical data rather than model complexity.

1 Introduction

Diabetes is a growing public health concern in the United States. Recent studies have shown that approximately 38.4 million Americans are living with diabetes [2]., while roughly 98 million adults have prediabetes and are at risk of developing type 2 diabetes. [2]. It is important to study and understand the factors behind diabetes as this chronic disease is a leading cause of kidney failure, heart disease, stroke, and vision loss. Additionally, diabetes not only affects the human body, but it can also be associated with immense economic costs in both medical spending and lost jobs or productivity. Thus, since many cases develop gradually and may remain undiagnosed for years, there is a high demand for tools that can flag risks in individuals as a form of early intervention.

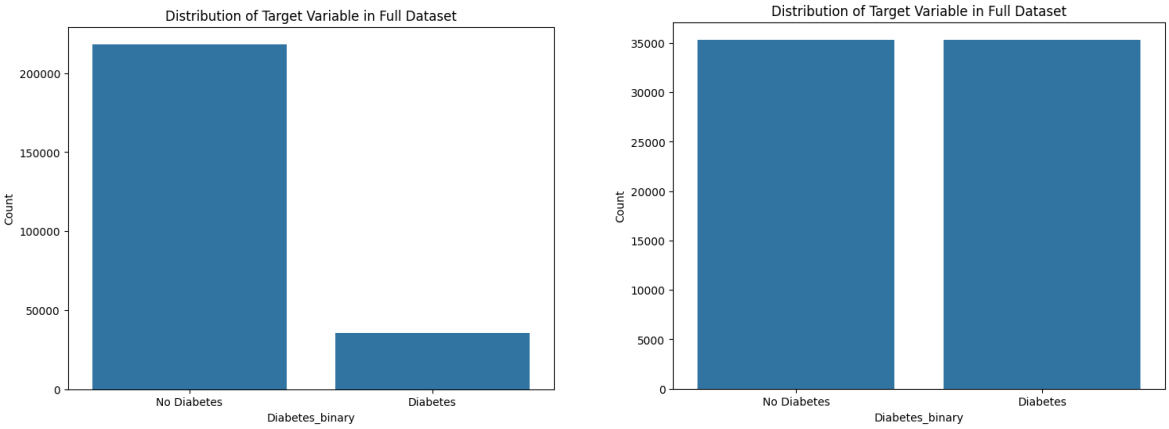
This project intends to test and understand the effectiveness of different supervised machine learning models in classifying whether an individual is likely to have diabetes or not based on their health and lifestyle-related features. By framing diabetes status as a binary classification problem, we are able to train machine learning models and compare their performance to determine which approaches are more effective for early risk prediction for later intervention.

2 Data

This paper utilizes a dataset provided by the UCI Machine Learning Repository titled "CDC Diabetes Health Indicators". The original dataset is derived from the CDC's Behavioral Risk Factor Surveillance System (BRFSS) and contains 253,680 adult survey responses. Each response has 21 health and lifestyle-related features along with a target label indicating diabetes status (i.e., no diabetes, prediabetes, or diabetes). The 21 features and their type (i.e., integer, binary) are as follows:

1. HighBP: High Blood Pressure (Binary)
2. HighChol: High Cholesterol (Binary)
3. CholCheck: Cholesterol Check (Binary)
4. BMI: Body Mass Index (Integer)
5. Smoker: Smoked at least 100 cigarettes in their lifetime (Binary)
6. Stroke: Ever had a stroke (Binary)
7. HeartDiseaseorAttack: History of heart disease or attacks (Binary)
8. PhysActivity: Exercise in the past 30 days (Binary)
9. Fruits: Eats fruit 1 or more times per day (Binary)
10. Veggies: Eats veggies 1 or more times per day (Binary)
11. HvyAlcoholConsump: 14 or 7 drinks per week based on sex (Binary)
12. AnyHealthcare: Current health care plan (Binary)
13. NoDocbcCost: Did not see a doctor in the past 12 months due to cost (Binary)
14. GenHlth: General Health on a scale of 1-5 (Integer)
15. MentalHlth: Number of days in the past 30 days when mental health was not good (Integer)
16. PhysHlth: Number of days in past 30 days when physical health was not good (Integer)
17. DiffWalk: Serious difficulty walking or climbing stairs (Binary)
18. Sex: Male or Female (Binary)
19. Age: Demographic (Integer)
20. Education: Scale 1-6, indicating years of education completed (Integer)
21. Income: Income category (1-8)

In the original dataset, the data is majorly imbalanced, with the vast majority of respondents reporting no diabetes, while only a small fraction reported pre-diabetes or diabetes. This means that training the models would skew results due to overfitting towards the majority class (non-diabetic), which in turn leads to a classifier achieving high accuracy by simply predicting "no diabetes" most of the time. See Figure 1a to understand the disparity.



(a) Original dataset

(b) Balanced dataset

Figure 1: Distribution of target variables in the original (left) and balanced (right) datasets.

To address this issue, I used a cleaned and balanced version of the CDC Health Indicators dataset that was originally published on Kaggle. This version of the dataset contains a total of 56,553 survey responses. It is important to note that the UCI Machine Learning Repository is not the source; rather, UCI is actually mirroring a version of the dataset found on Kaggle. The cleaned, balanced version I used was created by a data scientist named Alex Teboul, who sourced the original data from the BRFSS 2015 Dataset found on Kaggle. [1] See Figure 1b for the distribution of target variables in the balanced dataset.

To prepare the dataset for modeling, the balanced comma-separated value (CSV) file was first uploaded to Google Drive and loaded onto a Google Colab Notebook using pandas. A basic preprocessing step was performed to verify that there were no missing values. Additionally, an inspection was done to ensure that all target variables were encoded as 0/1. The last step to prepare the data for modeling was to randomly split the dataset into training and test sets using an 80/20 split. This is meant to preserve class balance in both subsets and is essentially a foundation for classification experiments. For models sensitive to feature scale, the continuous predictors were later standardized using a scaler (`StandardScaler` in scikit-learn).

3 Modeling

As mentioned before, this project portrays diabetes status as a binary classification problem, where for each individual in the dataset, the model predicts whether they are likely to have diabetes/pre-diabetes (1) or not (0) based on their 21 health and lifestyle features. Since this is a binary classification problem, we tested classification algorithms through six different types of models:

1. Logistic Regression (LR)
2. Decision Tree (DT)
3. Random Forests (RF)
4. Support Vector Machine (SVM)
5. K-Nearest Neighbors (KNN)
6. Multi-layer Perceptron (MLP)

Some of these models are sensitive to the scale of input features; thus, I applied a `StandardScaler`

to the training data and then performed a transformation on the test data. This configuration was used for logistic regression, SVM, KNN, and the MLP Classifier. On the other hand, tree-based models (i.e., decision tree, random forest) were trained on the original, unscaled features.

To better understand the subsequent code snippets, the following helper function was made to standardize the modeling procedure. The following function fits a given model on the training data, makes predictions on the test data, and reports accuracy and a full classification report containing the model's precision, recall, and the F1-score.

```
def evaluate_model(name, model, X_train, X_test, y_train, y_test):
  print(f'Evaluating {name}...')
  model.fit(X_train, y_train)
  y_pred = model.predict(X_test)
  accuracy = model.score(X_test, y_test)

  # store result
  model_results[name] = accuracy

  # print result
  print(f'{name} Accuracy: {accuracy}')
  print(classification_report(y_test, y_pred))

  return y_pred
```

3.1 Logistic Regression

The baseline for this project is logistic regression, a linear classification model that estimates the probability that an individual has diabetes (class 1) based on a weighted combination of their features. In this project, a logistic regression model was trained on the scaled training data using scikit-learn's `LogisticRegressor`. The model was fit with its default parameters, as adjusting them lowered its resulting metrics. The resulting logistic regression model resulted in a prediction accuracy of 74.58%, which acts as the comparable baseline for this project.

```
lreg = LogisticRegression()
y_pred_lreg = evaluate_model("Logistic Regression", lreg, X_train_scaled, X_test_scaled, y_train, y_test)
```

```
Evaluating Logistic Regression...
Logistic Regression Accuracy: 0.7458094631869298
              precision    recall  f1-score   support

         0.0       0.76      0.73      0.74      7070
         1.0       0.74      0.76      0.75      7069

    accuracy                           0.75     14139
   macro avg       0.75      0.75      0.75     14139
weighted avg       0.75      0.75      0.75     14139
```

3.2 Decision Trees

Decision trees are non-linear classification models that split a feature space into a series of decision rules. These rules can be interpreted in the form of an "if-then-else" block. In

this project, a decision tree classifier was trained on the unscaled data using scikit-learn's `DecisionTreeClassifier`. In addition, I implemented two versions of this model. One instance has a `max_depth` of 8, and the other has no maximum depth. The depth-limited tree is intended to reduce overfitting, but also potentially increase the performance metrics. Comparing the decision tree's performance to the logistic regression baseline helps show whether allowing non-linear splits and feature interactions improves on classification metrics.

```
# note: this dataset has major potential for overfitting due to the lower number of positive classifications.
# to counteract this, I set a max_depth of 8 for the following DecisionTreeClassifier to prevent overfitting.
# not setting a max_depth results in an accuracy of 0.651319046608671

dt = DecisionTreeClassifier(max_depth=8, random_state=42)
dt_no_limit = DecisionTreeClassifier(random_state=42)
y_pred_dt = evaluate_model("Decision Tree", dt, X_train, X_test, y_train, y_test)
y_pred_dt_no_limit = evaluate_model("Decision Tree (No Limit)", dt_no_limit, X_train, X_test, y_train, y_test)
```

```
Evaluating Decision Tree...
Decision Tree Accuracy: 0.737605205460075
              precision    recall  f1-score   support

         0.0       0.77      0.68      0.72      7070
         1.0       0.71      0.79      0.75      7069

    accuracy                           0.74     14139
   macro avg       0.74      0.74      0.74     14139
weighted avg       0.74      0.74      0.74     14139

Evaluating Decision Tree (No Limit)...
Decision Tree (No Limit) Accuracy: 0.651319046608671
              precision    recall  f1-score   support

         0.0       0.65      0.66      0.66      7070
         1.0       0.66      0.64      0.65      7069

    accuracy                           0.65     14139
   macro avg       0.65      0.65      0.65     14139
weighted avg       0.65      0.65      0.65     14139
```

As seen in the cell's terminal above, the decision tree with a `max_depth` of 8 resulted in an accuracy of $\tilde{7}3.76\%$ whereas the decision tree without a `max_depth` resulted in an accuracy of $\tilde{6}5.13\%$. This indicates that constraining the tree's depth improves the model's generalization. The very deep tree will overfit the training data while the shallower tree has a better prediction accuracy compared to both the unrestricted tree. However, despite being faster than the deeper tree, it does not beat the baseline of 74.58% imposed by the baseline's logistic regression.

3.3 Random Forests

Random forests are an extension of the decision trees algorithm, except that they create a "forest" of decision trees. Each tree is trained on a sample of the data, and at every split, it only considers a random subset of features. In this project, I used scikit-learn's `RandomForestClassifier` on the unscaled data, and tuned the `n_estimators` hyperparameter to test whether or not this parameter has a relationship with the prediction accuracy.

```
# most optimal from above
rf = RandomForestClassifier(n_estimators=5000, random_state=42, n_jobs=-1)
y_pred_rf = evaluate_model("Random Forest", rf, X_train, X_test, y_train, y_test)
```

```
Evaluating Random Forest...
Random Forest Accuracy: 0.737605205460075
              precision    recall  f1-score   support

         0.0       0.76      0.70      0.73      7070
         1.0       0.72      0.78      0.75      7069

    accuracy                           0.74     14139
   macro avg       0.74      0.74      0.74     14139
weighted avg       0.74      0.74      0.74     14139
```

For this model's implementation, I decided to adjust the number of trees contained in the forest by trying several values of `n_estimators` (i.e., 50, 100, 200, and 500) and recorded the resulting test accuracies in Table 1.

Number of estimators	Accuracy
10	0.7107999151283684
50	0.7323007284815051
100	0.7335738029563619
200	0.7358370464672184
500	0.7367564891435038
1000	0.7367564891435038
2000	0.7373223000212179
5000	0.7376052054600750
7000	0.7370393945823609
10000	0.7364028573449325

Table 1: Random Forest accuracy for different numbers of estimators

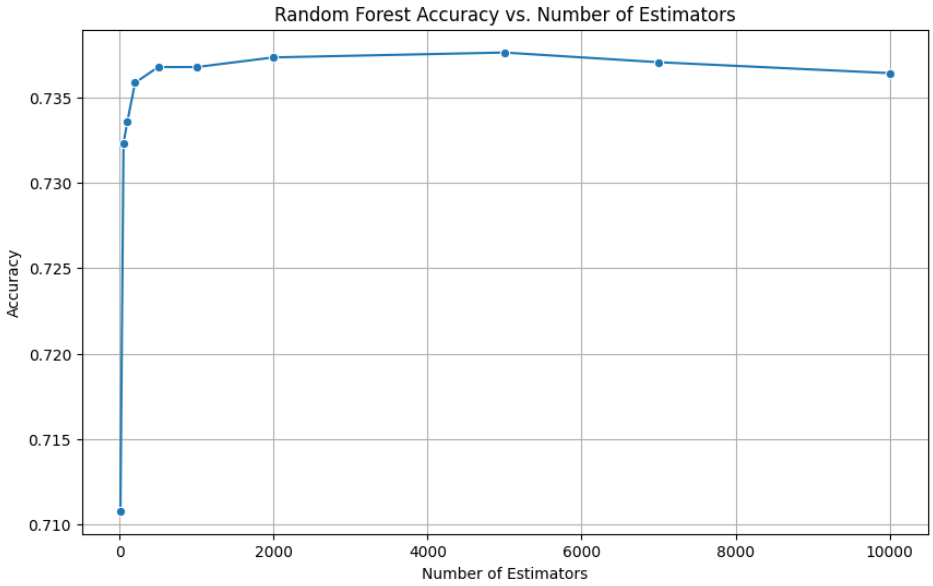


Figure 2: Graph of Random Forest Estimators vs Accuracy

Based on Figure 2 and Table 1's results, we can conclude that as the number of trees increased, the model's accuracy slightly improved but eventually diminished after testing 7000 trees. Since the accuracy decreased after the attempt with 7000 trees, the most optimal `n_estimators` value is likely between 5000 and 7000 trees. However, even at its best (5,000 trees, 73.76% accuracy), the random forest does not surpass the logistic regression baseline of 74.58%.

3.4 Support Vector Machines

Support Vector Machines (SVMs) are classifiers that maximize the distance between two classes (i.e., the margin). This essentially creates an optimal boundary to separate data points into different classes. In this project, I used scikit-learn's `LinearSVC` function, which implements a linear SVM with a "squared hinge" loss. A "squared hinge" loss is a differentiable loss function defined as the squared value of $max(0, 1 - y_{true} * y_{pred})$. This is often used in SVMs when performing binary classification. [3] Additionally, to accommodate SVM's sensitivity to scale features, the model was trained on the scaled dataset. The resulting `LinearSVC` Model achieved an accuracy of approximately 74.61% on the test set, which is slightly better than the baseline logistic regression model, which performed at 74.58%.

```
lsvc = LinearSVC(C=1, loss='squared_hinge', dual=False, random_state=42, max_iter=10000)
y_pred_lsvc = evaluate_model("LinearSVC", lsvc, X_train_scaled, X_test_scaled, y_train, y_test)

Evaluating LinearSVC...
LinearSVC Accuracy: 0.7460923686257869
              precision    recall  f1-score   support

         0.0       0.76      0.72      0.74      7070
         1.0       0.73      0.77      0.75      7069

    accuracy                           0.75     14139
   macro avg       0.75      0.75      0.75     14139
weighted avg       0.75      0.75      0.75     14139
```

3.5 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is another algorithm used for classification; however, it makes predictions by finding the "K" most similar data points which are termed "neighbors". I used scikit-learn's `KNeighborsClassifier` on the scaled dataset and tuned the `n_neighbors` hyperparameter to see how the number of neighbors affected prediction accuracy.

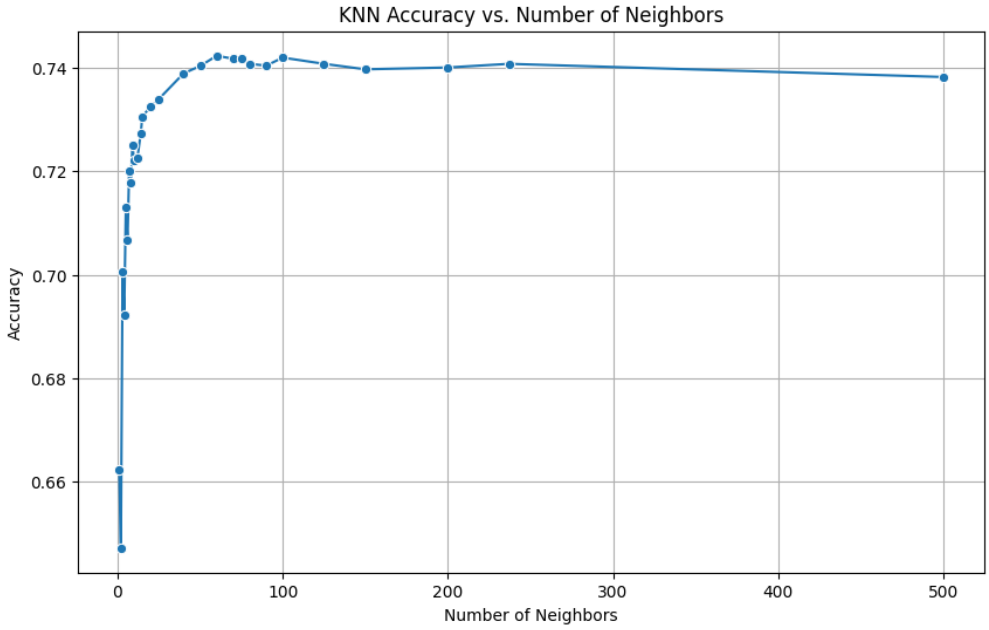


Figure 3: Graph of KNN Accuracy vs Number of Neighbors

Number of Neighbors (K)	Accuracy
1	0.6624230850838108
3	0.7006860456892283
5	0.7129924322795106
15	0.7306032958483627
25	0.7339981611146474
50	0.7403635334889314
75	0.7417780606832166
100	0.7419902397623594
125	0.7407878916472169
150	0.7397269962515030
237	0.7407878916472169

Table 2: KNN accuracy for different values of K

To do this, I tested several values of K (1,3,5,15, 25, 50, 75, 100, 125, 150, and 237). Then, I recorded the results in Figure 3 and Table 2 above. From first glance, it can be said that the model's accuracy increases steadily as `K` grows from 1 to 100 neighbors. However, beyond 100 neighbors is where accuracy dropped below 74.1%. 237 was chosen as an additional test value because it is common practice to take the square root of the number of test samples and use it as the number of neighbors. In this case, however, this value of `K` did not provide any improvement over the smaller values. Despite performing relatively well compared to the the tree models, the KNN model did not surpass the logistic regression baseline of 74.58%.

3.6 Multi-layer Perceptron Classifier

Multi-layer Perceptrons (MLPs) are a type of feed-forward neural network with three layers: an input layer, one or more hidden layers, and an output layer. It uses these layers to map the input features of the dataset to the output labels. In this project, I used scikit-learn's `MLPClassifier` with a single hidden layer, ReLU activations, the default Adam optimizer, and early stopping enabled. Additionally, this model was trained using the scaled dataset. To tune this model, I varied the hidden layer size (i.e., 250, 500, 1,000, 2,500, 5,000, 7,500, 10,000 neurons) and the validation fraction parameter. Specifically, I tested each hidden layer variation with a validation fraction of 0.1 and 0.2. The results can be found below in Figure 4, Table 3, and Table 4.

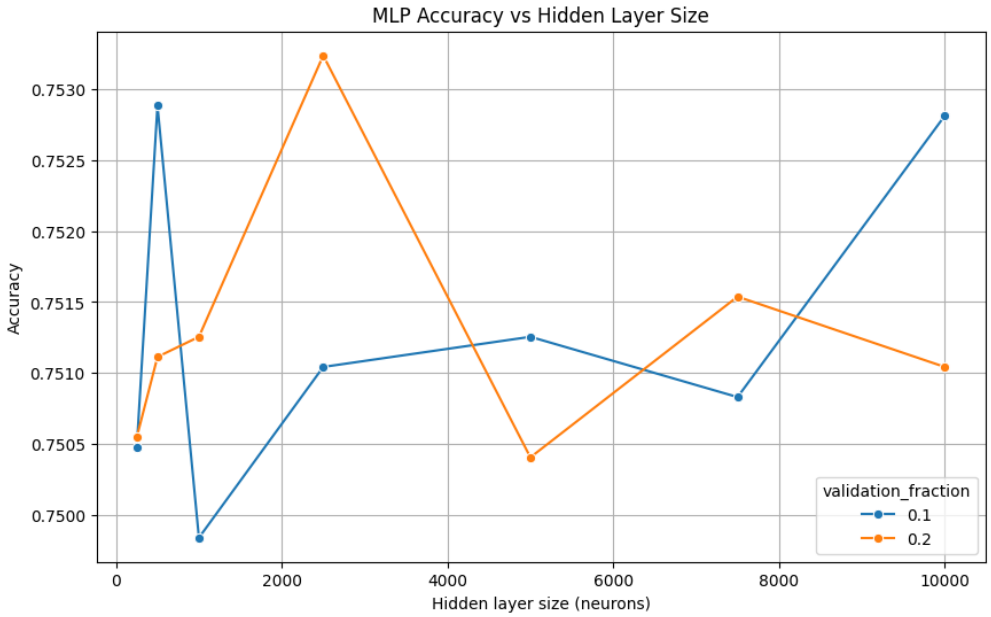


Figure 4: Graph of MLP Accuracy vs Hidden Layer Size (Varied by Validation Fraction)

Hidden Layer Size	Accuracy
250	0.7504774029280713
500	0.7528820991583564
1000	0.7498408656906429
2500	0.7510432138057854
5000	0.7512553928849283
7500	0.7508310347266426
10,000	0.7528113727986421

Table 3: VF = 0.1

Hidden Layer Size	Accuracy
250	0.7505481292877856
500	0.7511139401654997
1000	0.7512553928849283
2500	0.7532357309569276
5000	0.7504066765683570
7500	0.7515382983237853
10,000	0.7510432138057854

Table 4: VF = 0.2

As shown in Figure 4, both validation settings produce very similar accuracy curves; however, a validation fraction of 0.2 paired with a single hidden layer size of 2500 neurons produced the highest accuracy of 75.32% across all iterations. This makes MLP the best-performing model tested for this project. Additionally, this suggests that there is still room for improvement. It is possible that the most optimal hidden layer size is between 2500 and 5000 since accuracy decay occurred somewhere between these two points.

4 Results

Table 5 and Figure 5 show the test accuracies of all six models side by side when optimized to their best parameters tested. The accuracies range from roughly 73% to a little over 75%.

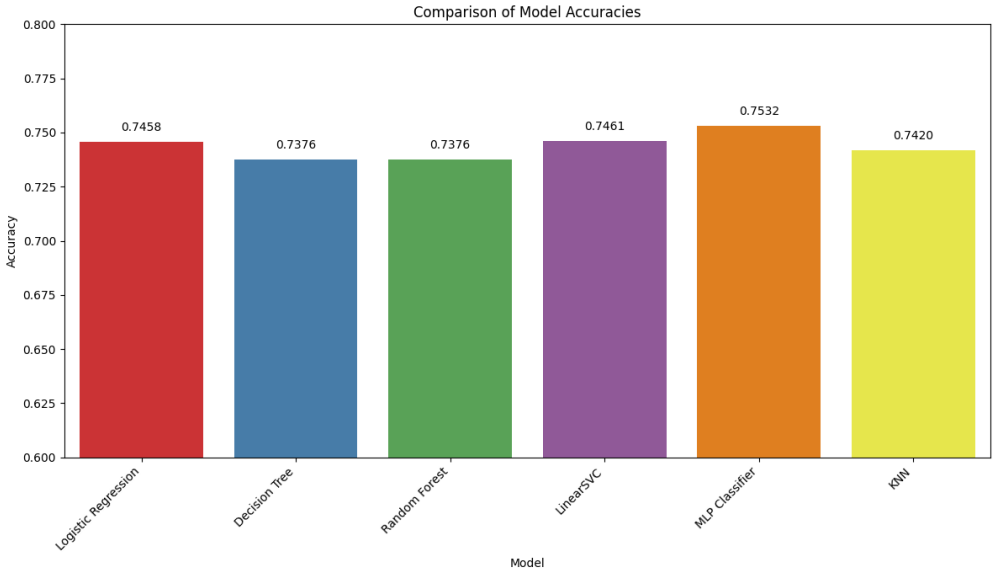


Figure 5: Comparison of Model Accuracies

Model	Accuracy
Logistic Regression (Baseline)	74.58%
Decision Tree	73.76%
Random Forest	73.76%
Linear SVM	74.61%
K-Nearest Neighbors	74.20%
MLP Classifier	75.32%

Table 5: Accuracy of different classification models

Figure 6 showcases the confusion matrices of all six models. This visualization helps illustrate how each one trades off false positives and false negatives.

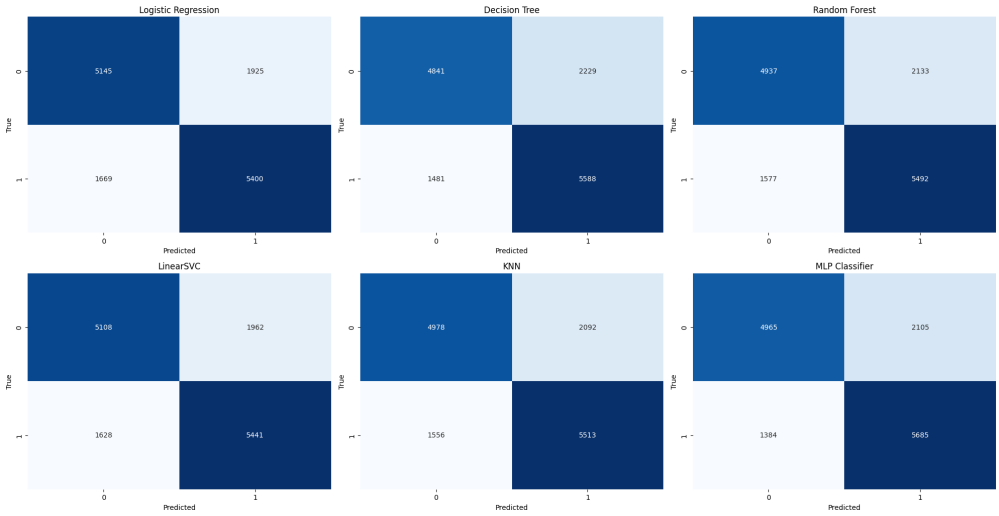


Figure 6: Comparison of Confusion Matrices of All Six Models

Figure 7 showcases the ROC curve of all six models, which plots the true positive rate against the false positive rate across different classification thresholds.

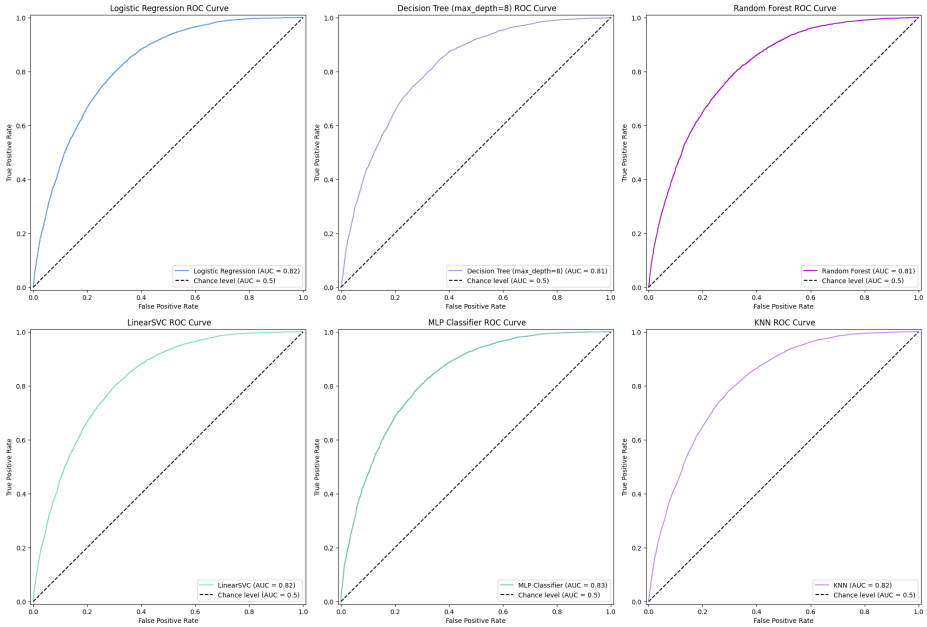


Figure 7: Comparison of ROC Curve of All Six Models

5 Discussion

When comparing the accuracies of all 6 models using Table 5 and Figure 5, we can see that performance is fairly similar, with all test accuracies being in between 73% and 75%. The best overall performance was obtained by the MLP Classifier, which reached an accuracy of 75.32%

with a single hidden layer of 2,500 neurons and a validation fraction of 0.2. Although this is the highest score among all the models, the improvement is scarce. Since the improvement is scarce, it is plausible to say that simpler methods like Logistic regression and SVM can perform almost on par with a more complex model like MLP in this scenario.

From Figure 6, one can see that among all the models, they consistently produce more false positives than false negatives, meaning they are more likely to incorrectly classify non-diabetic individuals as diabetic. The MLP classifier is the strongest in identifying positive cases as it has the highest number of true positives (5,685) while also having the lowest number of false negatives (1,384). It is important to note that this despite having the highest accuracy, this model still had the third highest number of false positives (2,105), meaning there is a slight trade-off in its precision.

From Figure 7, we can see that all curves lie well above the diagonal "chance" line provided by the `from_estimator` method from the scikit-learn `RocCurveDisplay` object. This indicates that every model performs substantially better than random guessing (i.e., 50%), with the MLP Classifier achieving the highest overall area under the curve.

6 Conclusion

In summary, the purpose of this work was to investigate how different types of machine learning problems perform on a binary classification problem. More importantly, it was used to see which model could correctly predict diabetes based on risk factors.

One conclusion that can be drawn from this study is that most standard supervised models perform similarly on this dataset. All six models achieved test accuracies in very similar ranges after optimizing their hyperparameters, suggesting that the available features limit how much performance can be improved, regardless of the modeling approach.

Another conclusion that can be drawn is that the arguably highest computational model, the MLP classifier, offers only a slight advantage over the less computationally expensive models like Linear SVM and Linear Regression. While the MLP achieved the highest accuracy and AUC, it gained under one percentage point and came with a small increase in false positives, indicating a trade-off between detecting more diabetic individuals and maintaining high precision.

In future work, this project could be developed to incorporate larger clinical data. It is important to note that before sticking with the balanced version of the dataset, I initially worked with the raw dataset and attempted to balance it using SMOTE. However, it proved difficult to optimize and often failed to improve performance consistently, leading to the final analysis being concentrated on the balanced dataset. Thus, I believe a larger and balanced dataset that encompasses a wide range and consistent pattern of the features would significantly strengthen the reliability and usefulness of the model. At the moment, these models can guess better than random guessing (i.e., over 50%); however, these accuracies can be considered mediocre, and there is more room for improvement. A larger, well-balanced dataset will allow more complex models like MLP to potentially perform better, so they can be worth the computational cost. This, in turn, could lead to more engineering, more accessible tools for early risk identification for intervention, as I had mentioned at the beginning of this paper.

References

[1] Alex Teboul. *Diabetes Health Indicators Dataset* [Data set]. 2021. Kaggle. https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset?select=diabetes_binary_5050split_health_indicators_BRFSS2015.csv. Accessed December 2, 2025.

[2] American Diabetes Association. *Statistics About Diabetes*. 2023. https://diabetes.org/about-diabetes/statistics/about-diabetes. Accessed December 3, 2025.

[3] Haibal. *Sqaured Hinge Loss Metric Documentation*. 2025. https://haibal.com/documentation/metric-squared-hinge/ Accessed December 2, 2025.