Supervised Machine Learning: Classification

Report

Lucas Mogg
06/02/2024

All code and data are available at Kaggle.com (link)

Introduction and Main Objective

The main objective of the analysis is to model the markers to predict and identify likely diabetes cases within a Pima Indian community from biological indicators and measurement features. Whilst prediction performance is important (using a metric such as precision), the strongest indicators would preferably be interpretable to have significance in understanding potential correlation.

Dataset Description

The dataset chosen here is a Pima Indians Diabetes Database, from kaggle (link). It contains the following columns of data on individual Pima Indian individuals:

Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | Diabetes Pedigree Function | Age | Outcome (1: diabetic 0: non-diabetic)

Most columns are self explanatory, but the Diabetes Pedigree Function can be described as: a function of the diabetes history in relatives and the genetic relationship of those relatives to the subject. The outcome variable (target variable) is a binary outcome of either 1 or 0 (1: diabetic 0: non-diabetic) .

Outline of objective from analysis

The plan of this analysis is to predict the target outcome variable from the other features in the dataset. It should identify the biggest risk factors that are likely to correlate to a positive diabetes outcome.

Summary of data exploration: actions taken for data cleaning and feature engineering.

A description of the dataset and feature types are provided in the following tables. No encoding is necessary as all values are either int64 or float64. A check is performed to see if there are any missing values, using the df.isnull().sum() method, where no missing values are found[1]. A visualization of the counts for the outcome variable is also provided below, where we can see that the dataset is unbalanced, with around ⅓ of the outcome classes being 1 (positive diagnosis of diabetes) and ⅔ of the outcome classes belonging to 0 (negative diagnosis of diabetes). The approach to handle the imbalanced dataset will be to utilise class weighting for the outcome variable. Oversampling and undersampling will also be explored for the best performing model.

[1] Although later in the work it was found that missing values of skin thickness and insulin are encoded as zeros.

Feature Scaling

The spread in data mean values (incorporating standard deviation) spans two orders of magnitude (as can be seen in the following tables). In particular, the DiabetesPedigreeFunction mean value is at least an order of magnitude smaller than other features, such as blood glucose level. Therefore, scaling would be beneficial for models where regularisation and distance between datapoints is important to a model (such as support vector machines SVM and k-nearest neighbours k-NN).

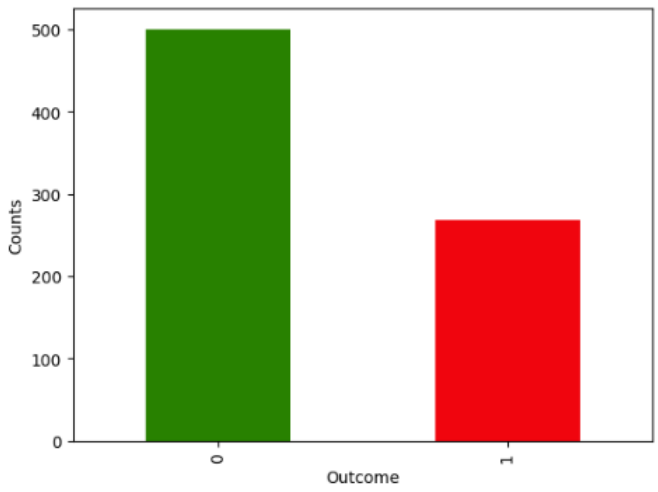
Left: Table X - describe() function for the dataframe. Right: Table of unique classes for each feature in the dataframe.

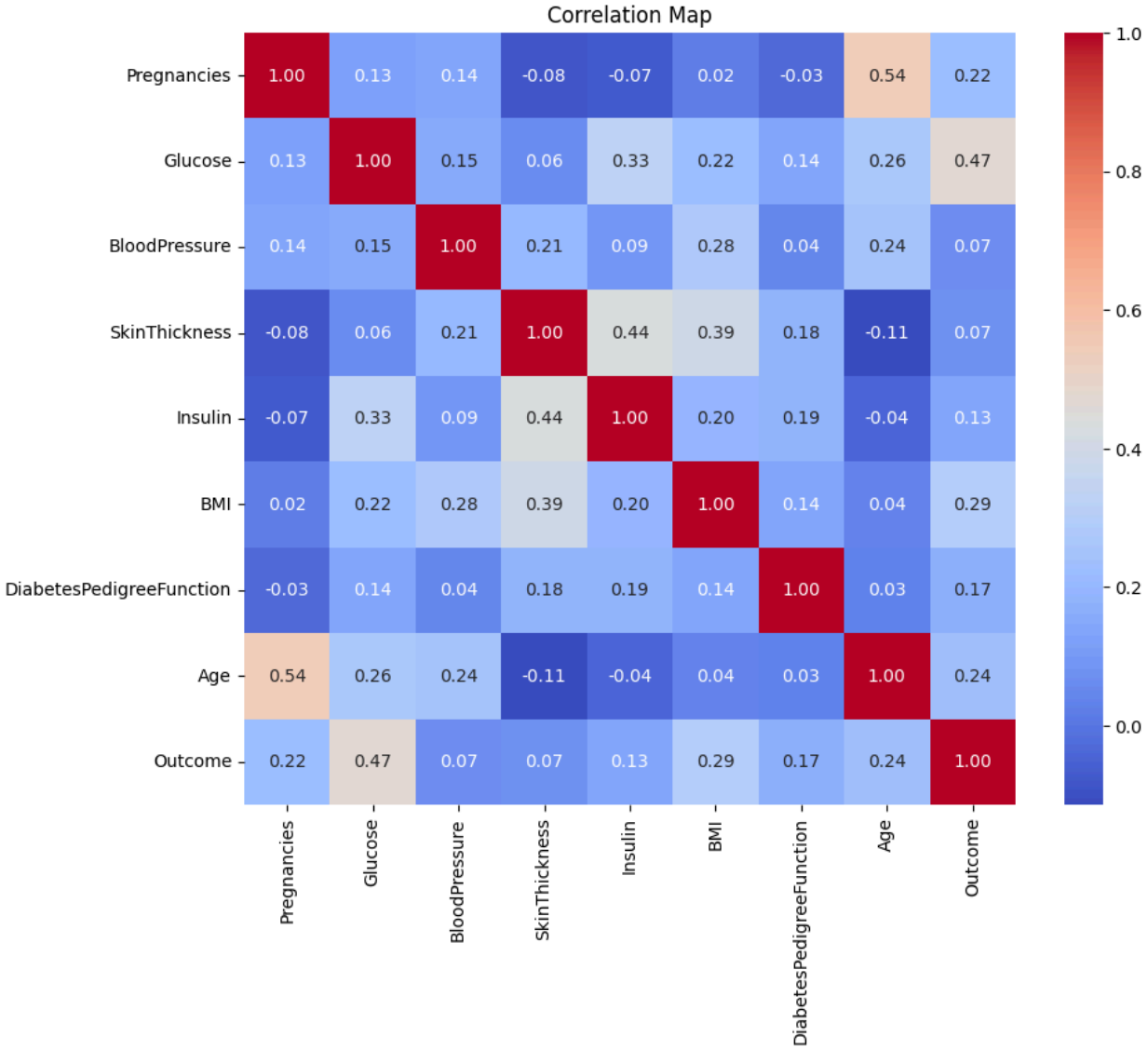
	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetes Pedigree Function	Age	Outcome
count	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00
mean	3.85	120.89	69.11	20.54	79.80	31.99	0.47	33.24	0.35
std	3.37	31.97	19.36	15.95	115.24	7.88	0.33	11.76	0.48
min	0.00	0.00	0.00	0.00	0.00	0.00	0.08	21.00	0.00
25%	1.00	99.00	62.00	0.00	0.00	27.30	0.24	24.00	0.00
50%	3.00	117.00	72.00	23.00	30.50	32.00	0.37	29.00	0.00
75%	6.00	140.25	80.00	32.00	127.25	36.60	0.63	41.00	1.00
max	17.00	199.00	122.00	99.00	846.00	67.10	2.42	81.00	1.00

	Unique Classes
Pregnancies	17
Glucose	136
BloodPressure	47
SkinThickness	51
Insulin	186
BMI	248
DiabetesPedigreeFunction	517
Age	52
Outcome	2

Left: Table X - dtypes() function for the dataset. Right: Visualization of count of outcome variable.

Pregnancies	int64
Glucose	int64
BloodPressure	int64
SkinThickness	int64
Insulin	int64
BMI	float64
DiabetesPedigreeFunction	float64
Age	int64
Outcome	int64





Feature and Outcome Correlation Map

The correlation map quickly highlights a couple of, perhaps expected, correlations between glucose and BMI with a positive outcome fpr diabetes, 0.47 and 0.29, respecitvely. Pregnancy and age also correlate positively to a diabetic outcome at 0.22 and 0.24, respectively. There are also some other slighly unexpected cross feature correlations such as skin thickness and Insulin at 0.44. Age and pregnancy number are expectledy positively correlated at 0.54.

Models trained

All models, except for the baseline 'vanilla' Logistic Regression[2] are trained using a *pipeline* and *parameter_grid* passed into a *GridSearchCV*. This search contains a cross validation value of 5, meaning 5 k-fold splits are performed, verbosity of 1 and is scored using the f1 metric (to try and accommodate the unbalanced nature of this dataset). Models are evaluated using the same standard model performance method, provided in appendix 1. Lastly, all models are trained on the same MinMax scaled feature matrix (except for the 'vanilla' Logistic Regression, which is unscaled data). The next sections present model performance for each different approach.

Logistic Regression

Logistic Regression is a statistical method for analysing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). It estimates the probability of an event occurring by fitting data to a logistic function, providing a binary outcome (1/0, Yes/No). It is widely used for binary classification problems in various fields.

The logistic (sigmoid) function is:

$$f(x) = \frac{1}{1 + e^{-x}}$$

The linear combination x of input features (X) in logistic regression:

$$x = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

Where

$f(x)$ represents the output of the logistic function.
x is the linear combination of features.
$w_1, w_2, w_3 \ldots w_n$ are the weights assigned to each feature.
$x_1, x_2, \ldots x_n$ are the feature values for a given data point.
b is the bias term, an additional parameter used to adjust the output along the y-axis.

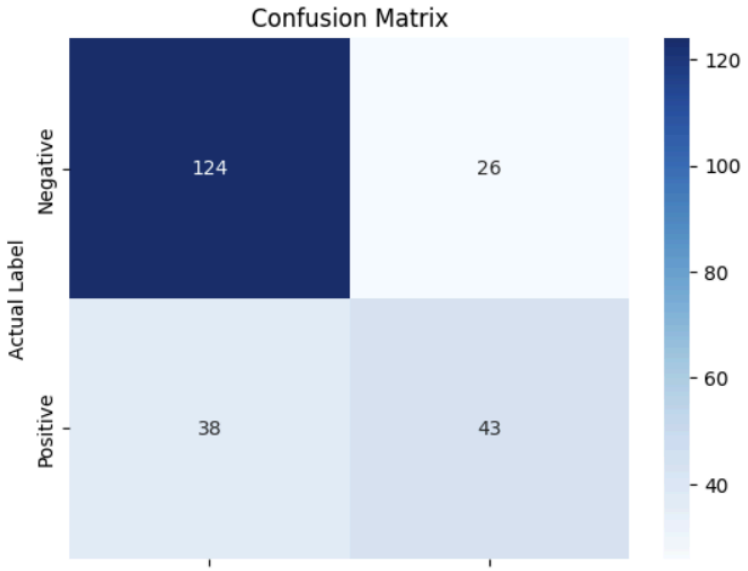
[2] Which lacks any regularisation or hyperparameter tuning.

‘Vanilla’ Logistic Regression

This is the simplest model to train, and will be used as a baseline. The performance metrics are summarised in the following table and confusion matrix.
After performing a GridSearch (code provided in appendix 2), the performance metrics are.

Logistic Regression with GridSearchCV Performance Output

‘Vanilla’ Logistic Regression Performance Output

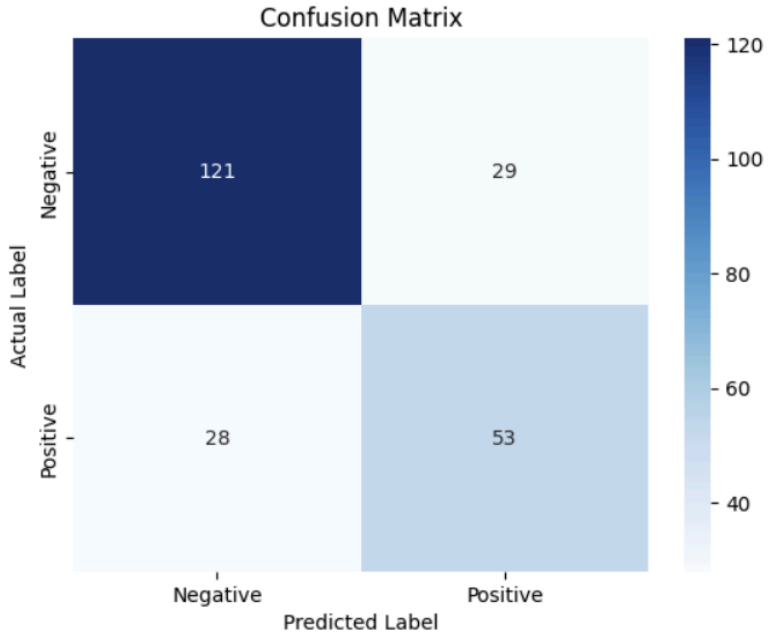


	model	precision	recall	fscore	accuracy	auc
0	Vanilla Logistic Regression	0.716	0.723	0.717	0.723	0.679

Logistic Regression with GridSearchCV optimised hyperparameters

The best found hyperparameters were:

```
{'model__C': 11.29,
 'model__class_weight': {0: 0.46, 1: 0.54},
 'model__penalty': 'l1',
 'model__solver': 'saga',
 'polynomial__degree': 1}
```



	model	precision	recall	fscore	accuracy	auc
0	Logistic Regression with hyperparameter tuning	0.754	0.753	0.754	0.753	0.73

The Logistic Regression with tuned hyperparameters displays an improved performance in terms of the metrics: model, precision, recall, fscore, accuracy and auc, as can be seen from the two performance outputs. There is a difference between the metrics reported in tables and those that would be calculated from the confusion matrices. This difference can arise from the model applying weighting, such as class weighting. Noticing that all of the metrics (to 3 significant figures) are the same, except for the area under the curve. This seems unlikely and the reasons for this are not clear.

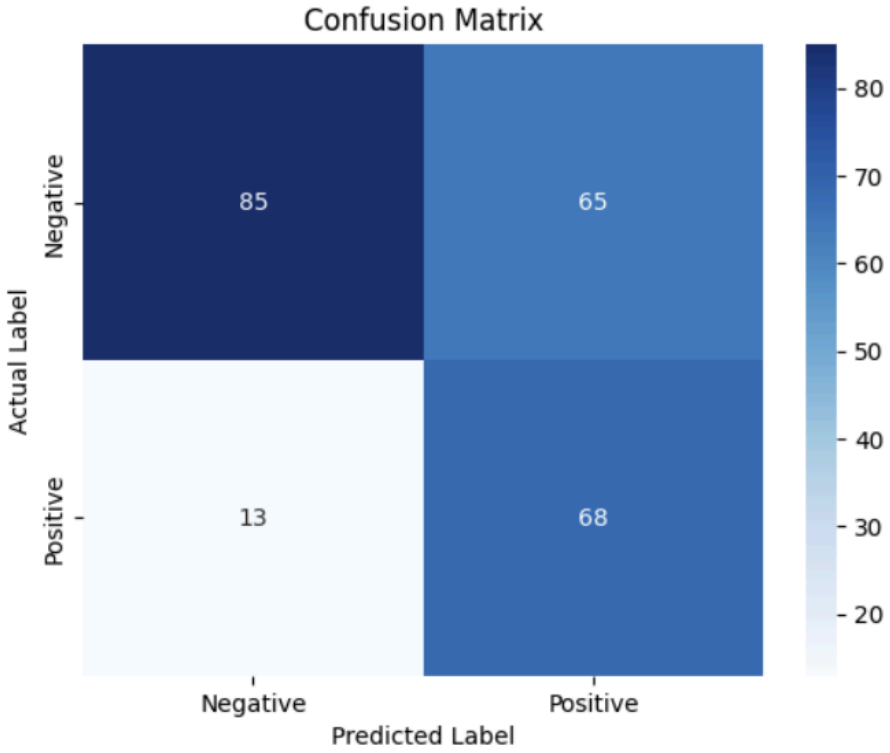
Decision Trees

Decision Trees are a non-linear predictive modelling tool. They represent decisions and their possible consequences as a tree-like model of choices. Each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label or a decision taken after computing all attributes. They are simple to understand and interpret but can be prone to overfitting.

To obtain the max depth number for this model to be searched over, the model is first allowed to run without any restriction. This will let the model train leaf nodes to contain to depths where no splits occur, allowing leaf nodes to be pure. After running this model for training data, the .max_depth attribute is 12. Therefore, GridSearchCV is searched over a range up until a depth of 12. The best parameter values were found to be:

```
{'model__class_weight': {0: 0.2777777777777778, 1: 0.7222222222222222},
                         'model__max_depth': 3,
                        'model__max_features': 6}
```

The performance metrics were measured as:



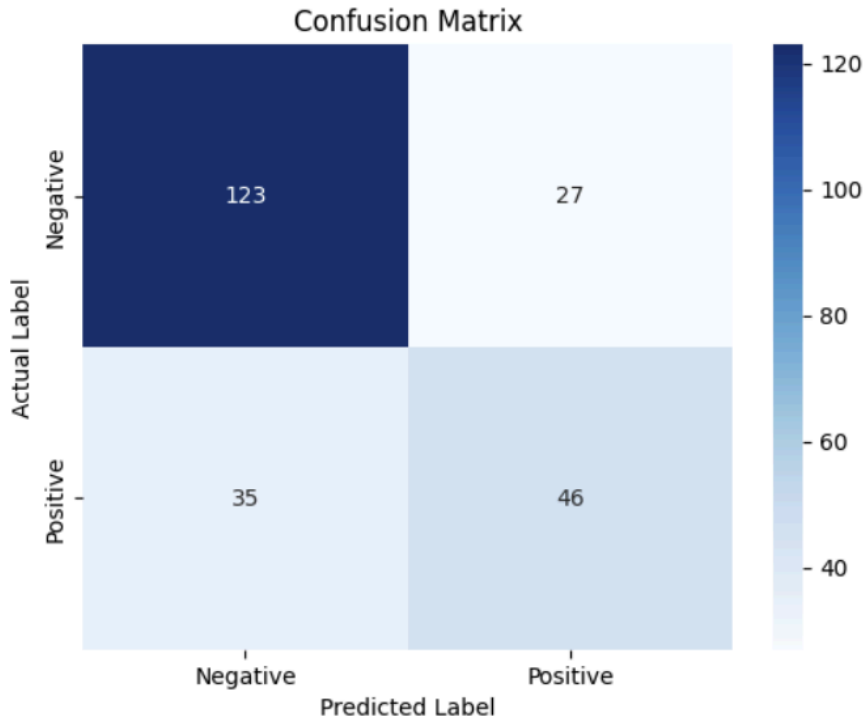
	model	precision	recall	fscore	accuracy	auc
0		0.742	0.662	0.668	0.662	0.703

Random Forest

Random Forests are an ensemble learning method primarily used for classification and regression tasks. They operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (for classification) or mean prediction (for regression) of the individual trees. Random Forests correct for decision trees' habit of overfitting to their training set by introducing randomness in two ways: each tree is built from a random sample of the data, and only a random subset of features is considered for splitting at each node. This results in a model that is robust, versatile, and performs well on many different types of data, though at the cost of some interpretability compared to single decision trees.

For this project, the Random Forest model was hyperparameter tuned using the GridSearchCV, as described in Appendix 4. The best parameter values were found to be:

```
{'model__max_depth': 5,
 'model__max_features': 'log2',
 'model__n_estimators': 15}
```



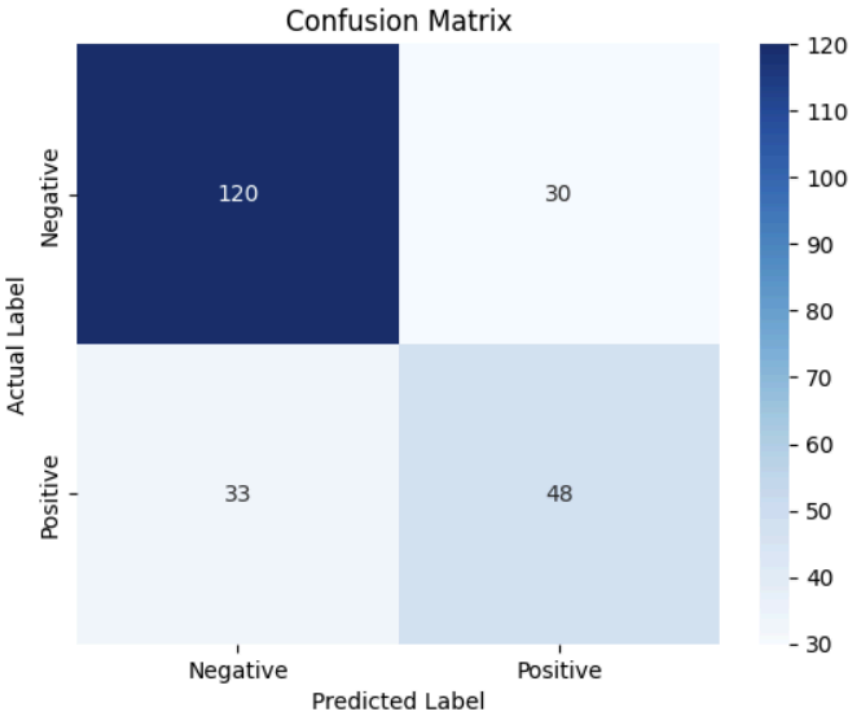
	model	precision	recall	fscore	accuracy	auc
0	Random Forest	0.726	0.732	0.728	0.732	0.694

K-Nearest Neighbors (KNN)

KNN is a simple, instance-based learning algorithm used in classification and regression. It operates by finding the 'k' training examples closest in distance to a given query point and predicting the output based on these nearest neighbours. KNN is non-parametric, meaning it makes no underlying assumptions about the distribution of data, making it versatile but also sensitive to the local structure of the data.

The fitted hyperparameters using GridsearchCV and assessment of model performance are provided below:

```
{'knnn__metric': 'minkowski', 'knnn__n_neighbors': 5}
```



	model	precision	recall	fscore	accuracy	auc
0		0.725	0.727	0.726	0.727	0.696

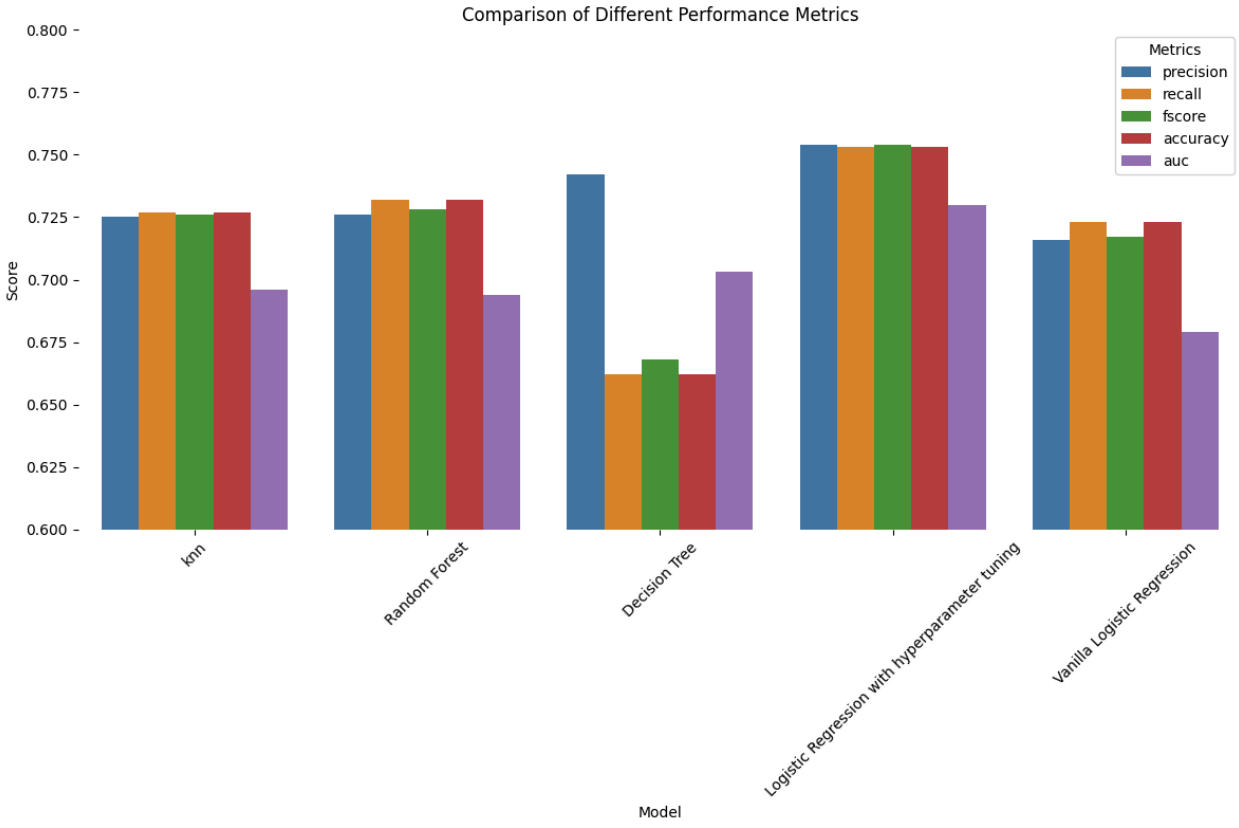
Summary of Key Findings and Insights

- A paragraph explaining which of your classifier models you recommend as a final model that best fits your needs in terms of accuracy and explainability.

Performance Metric Comparison

A comparison of all of the performance metrics are presented in the table below. All metrics are compared using bar charts, and presented in Appendix 5. The model comparisons for two metrics fscore and precision are provided in this section. We can see that Logistic Regression (with hyperparamter tuning performs best in terms of fscore and precision, indeed across all metrics.

model	precision	recall	fscore	accuracy	auc
knn	0.725	0.727	0.726	0.727	0.696
Random Forest	0.726	0.732	0.728	0.732	0.694
Decision Tree	0.742	0.662	0.668	0.662	0.703
Logistic Regression with hyperparameter tuning	0.754	0.753	0.754	0.753	0.730
Vanilla Logistic Regression	0.716	0.723	0.717	0.723	0.679



Insights from Model Interpretation

Interpretation of the Optimised Logistic Regression Model

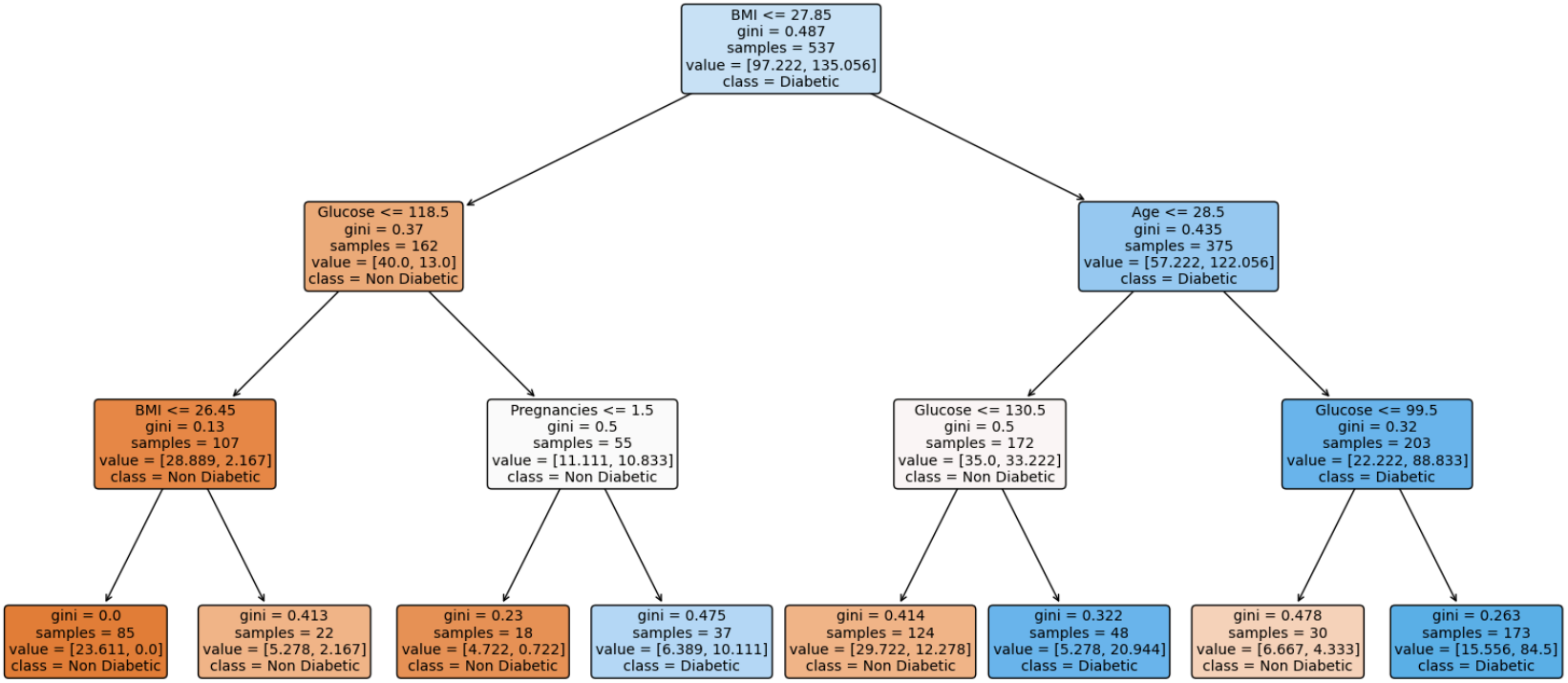
Below is the extracted feature coefficients for the optimised logistic regression model. These values are extracted from the GridSearchCV object using the method presented in the appendix section.

Feature	Coefficient
BMI	6.465194
Glucose	5.965271
Pregnancies	2.457134
DiabetesPedigreeFunction	2.378707
Age	1.106022
Insulin	-0.03159
SkinThickness	-0.861
BloodPressure	-1.31612

The model can therefore be interpretted to show that body mass index and blood glucose levels are the most significant positively correlated coefficients to a positive case of onset diabetes for this Pima Indian dataset. A larger positive coefficient indicates a stronger positive correlation between the corresponding feature and the likelihood of the positive class. It means that as the feature value increases, the probability of the target being the positive class increases. A larger negative coefficient indicates a stronger negative correlation, meaning that as the feature value increases, the probability of the target being the positive class decreases. The larger negative coefficient for blood pressure indicates such a negative correlation, when considering this Pima Indian dataset.

Interpretation of the Decision Tree Model

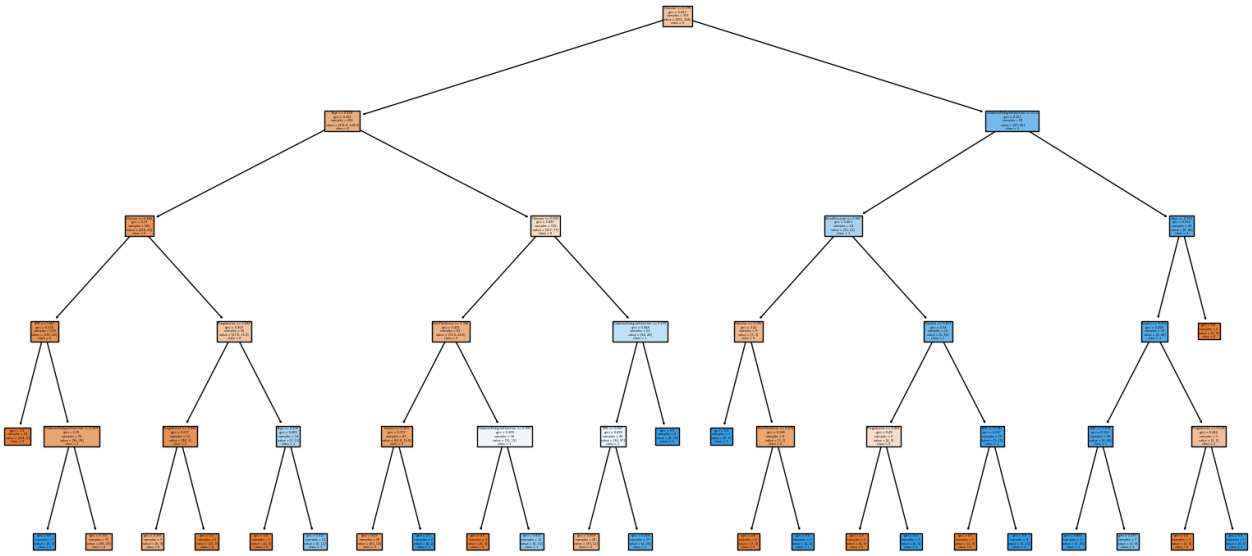
Using sklearn plot_tree function, the visualisation of the decision tree is plotted below. Unscaled data is used to train the model to enhance interpretability. The root node separates out data first by BMI, indicating that this is the most significant factor (as Decision trees use a greedy search). For the decision tree from scikit-learn, which follow the CART (Classification and Regression Trees) algorithm, the convention is that if the condition is met (e.g., if the feature's value is less than or equal to the threshold), the left branch is taken; if the condition is not met (the feature's value is more than the threshold), the right branch is taken. Next, data is split by Glucose on the left and Age on the right. Finally, further splits are made at layer 2 based on BMI, Pregnancy and Glucose.



As splits are only made on BMI, Glucose, Age and Pregnancy, this model has placed the most significance on these features. This is consistent with the Logistic Regression placing most significance on these features, except that Age has been used as a split rather than the DiabetesPedigreeFunction, which had a larger positive coefficient than age in Linear Regression.

Interpretation of the Random Forest Model

This was attempted by first identifying the number of trees that the RandomForestClassifier fits the model to, this was extracted to be 15 (see appendix section for the code to perform this). There are 15 trees used in the best_estimator model and each of these have depths of 5. This makes interpretation challenging. Below is an image of one of the 15 trees showing the complexity. It is, in principle, possible to interpret each tree and also investigate the tree voting for each row of the dataset to interpret the model output, but this is not performed here. Gaining insight from such an effort would be challenging.



Interpretation of k-NN model

For a given prediction, examining the nearest neighbors and their feature values can offer some insights into what characteristics are shared among neighbors. This can help infer which features might be influencing the grouping, albeit in a more qualitative and less direct way. However, having uncorrellated features may throw the effectiveness of this technique off.

Further, as the feature dimension for this dataset is 8 (not including the target), thus interpreting the nearest neighbours in this fashion is complex, along with the potential challenges of incorporating Minkowski distance. Nonetheless, as an exercise, nearest neighbours have been extracted using kneighbours, using the code snippet in the appendix section.

Upon inspecting the nearest neighbour data, it is apparent that skin thickness and insulin have many zero values. This is likely missing data values encoded as zeroes and this may have adverse impact on the modelling. No

See next page for nearest neighbour extraction.

Summary of nearest neighbours: Indexes, distances and feature values

Grey = test point
Red = Incorrectly classified near neighbour
Green = Correctly classified near neighbour

Nearest Neighbors' Indices: [[109 472 486 300 484]]
Nearest Neighbors' Distances: [[0.11467768 0.17908725 0.18480498 0.20094804 0.20381582]]

Test Point:
Pregnancies 0.000000
Glucose 0.663317
BloodPressure 0.639344
SkinThickness 0.000000
Insulin 0.000000
BMI 0.545455
DiabetesPedigreeFunction 0.131906
Age 0.000000
Name: 0, dtype: float64, Outcome Label: 0

Neighbor 1
, Index:
109, Features:
Pregnancies 0.000000
Glucose 0.567839
BloodPressure 0.622951
SkinThickness 0.000000
Insulin 0.000000
BMI 0.560606
DiabetesPedigreeFunction 0.082655
Age 0.033333
Name: 124, dtype: float64, Outcome Label: 1

Neighbor 2
, Index:
472, Features:
Pregnancies 0.000000
Glucose 0.733668
BloodPressure 0.573770
SkinThickness 0.000000
Insulin 0.000000
BMI 0.638047
DiabetesPedigreeFunction 0.106638
Age 0.116667
Name: 280, dtype: float64, Outcome Label: 1

Neighbor 3
, Index:
486, Features:
Pregnancies 0.117647
Glucose 0.648241
BloodPressure 0.688525
SkinThickness 0.000000
Insulin 0.000000
BMI 0.471380
DiabetesPedigreeFunction 0.085225
Age 0.100000
Name: 251, dtype: float64, Outcome Label: 0

Neighbor 4
, Index:
300, Features:
Pregnancies 0.058824
Glucose 0.693467
BloodPressure 0.672131
SkinThickness 0.000000
Insulin 0.000000
BMI 0.675084
DiabetesPedigreeFunction 0.064668
Age 0.116667
Name: 201, dtype: float64, Outcome Label: 0

Neighbor 5
, Index:
484, Features:
Pregnancies 0.000000
Glucose 0.507538
BloodPressure 0.622951
SkinThickness 0.000000
Insulin 0.000000
BMI 0.601010
DiabetesPedigreeFunction 0.048394
Age 0.083333
Name: 226, dtype: float64, Outcome Label:

Suggestions for next steps in analyzing this data,

To better understand the features that contribute to a positive diabetes diagnosis, dimensional reduction (using a technique such as principal component analysis - PCA) could make interpretation in some models (es[ecially k-NN) simpler. Findings from one model (such as feature importance assesment) could be fed into another model, such as k-NN and random forest by way of feature weighting.

Adding in time series data that tracks BMI and glucose history may add predictive power. For example, a person who has had a longer history (or fraction of lifespan) at elevated BMI may be more predisposed to diabetes.

To address the zero encoded values for missing values:

Data Exploration and Preprocessing: If these zeros represent missing data, imputation strategies could be used to replace zeros with meaningful statistics (mean, median, mode) or model-based imputation. It is not possible for these values to truely be zero, except for Pregnancies and perhaps DiabetesPedigreeFunction. For cases where insulin or Skin Thickness does not vary much across the dataset or contains a large proportion of zeros, it might be less informative for the model

Model Evaluation: Evaluate the performance of the k-NN model and consider whether excluding or transforming certain features could improve model interpretability and performance. This might involve running experiments with and without these features or applying different data preprocessing strategies to see how they impact model outcomes.

Appendices

Appendix 1: Standardised method to evaluate model performance

```
LR2 =
LogisticRegression(solver='liblinear',C=1.0,class_weight={0:0.5,1:0.5})

pipeline_LR = Pipeline([
    ('polynomial', PolynomialFeatures(include_bias=False, degree=2)),
    ('mm', MinMaxScaler()),
    ('model', LR2)])

params_grid = {
    'polynomial__degree': [1, 2, 3],
    'model__C': np.logspace(-4, 4, 20),  # Regularization strength
(inverse)
    'model__solver': ['lbfgs','saga'],
    'model__class_weight': [{0: x, 1: 1.0-x} for x in np.linspace(0.1, 0.9,
10)],  # Class weights for class 0 and 1
    'model__penalty': ['l1', 'l2']
}
search = GridSearchCV(pipeline_LR, param_grid=params_grid, scoring = 'f1',
verbose=1, cv= 5)
```

Appendix 2 Logistic Regression with tuned hyperparameters and cross validation.

```
LR2 =
LogisticRegression(solver='liblinear',C=1.0,class_weight={0:0.5,1:0.5})

pipeline_LR = Pipeline([
    ('polynomial', PolynomialFeatures(include_bias=False, degree=2)),
    ('mm', MinMaxScaler()),
    ('model', LR2)])

params_grid = {
    'polynomial__degree': [1, 2, 3],
    'model__C': np.logspace(-4, 4, 20),  # Regularization strength
(inverse)
```

```
    'model__solver': ['lbfgs','saga'],
    'model__class_weight': [{0: x, 1: 1.0-x} for x in np.linspace(0.1, 0.9,
10)],  # Class weights for class 0 and 1
    'model__penalty': ['l1', 'l2']
}
search = GridSearchCV(pipeline_LR, param_grid=params_grid, scoring = 'f1',
verbose=1, cv= 5)
```

Appendix 3 Decision Tree with tuned hyperparameters and cross validation.

```
pipeline = Pipeline([('scaler',MinMaxScaler()),('model',dt)])


param_grid = {'model__max_depth':range(1, dt.tree_.max_depth+1, 2),
              'model__max_features': range(1,
len(dt.feature_importances_)+1),
             'model__class_weight':[{0: x, 1: 1.0-x} for x in
np.linspace(0.1, 0.9, 10)]}

GR = GridSearchCV(pipeline,
                  param_grid=param_grid,
                  scoring='f1',
                  n_jobs=-1, cv= 5)

GR = GR.fit(X_train, y_train)
```

Appendix 4 Random Forest with tuned hyperparameters and cross validation.

```
# The decision tree classifier needs to be run before random forest is.
This is to extract max depth for when pure leaf nodes are reached

rf = RandomForestClassifier(random_state=142)

pipeline = Pipeline([('scaler',MinMaxScaler()),('model',rf)])
```

```
# The max features is often set as floor(sqrt(total number of features))
max_features = round(np.sqrt(X.shape[1]))-1

param_grid = {'model__max_depth':range(1, dt.tree_.max_depth+1, 2),
              # This will select the sqrt and other approaches to max
feature number
              'model__max_features':["auto", "sqrt", "log2"],
              #20 below is a bit arbitrary, default is 100
              'model__n_estimators':list(range(1,20,1))}

GR = GridSearchCV(pipeline,
                  param_grid=param_grid,
                  scoring='f1',
                  n_jobs=-1, cv= 5)

GR = GR.fit(X_train, y_train)
```

Appendix 5 Method to extract the coefficients for a Logistic Regression Model (Hyperparamter searched using GridSeacrhCV).

```
# Method to extract the coefficients
best_model = search.best_estimator_
logistic_regression_model = best_model.named_steps['model']
coefficients = logistic_regression_model.coef_
intercept = logistic_regression_model.intercept_

feature_names = X_train.columns  # Assuming X_train is a DataFrame

# Pair the feature names with the coefficients
coef_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients.flatten()})

coef_df = coef_df.sort_values(by = 'Coefficient', ascending = False)
sorted_df_reset = coef_df.reset_index(drop=True)

print(sorted_df_reset)
```

Appendix 6 Code to extract the number of trees from a grid search optimised RandomForestClassifier.

```
# Access the best RandomForest model
best_rf_model = GR.best_estimator_
rf_model = best_rf_model.named_steps['model']

# Print the number of trees
print(f"Number of trees in the optimized RandomForest model: {rf_model.n_estimators}")
```

Appendix 7 Code used to extract the nearest neighbours for a random X_test data point.

```
#Interpreting the k-NN model using kneighbours

# Isolate the k-NN model from the grid search

knn_estimator = Grid_knn.best_estimator_
knn_model = knn_estimator.named_steps['knnn']


# Next, make sure that the data is scaled in the same way
scaler = pipeline_knn.named_steps['scaler']
scaler.fit(X_train)

# Transform a specific row - note that you must ensure the input is 2D
(hence the reshaping)
X_test_sample = scaler.transform(X_test[3:4])
X_train_scaled_array = scaler.transform(X_train)
X_train_scaled = pd.DataFrame(X_train_scaled_array, index=X_train.index,
columns=X_train.columns)

distances, indices = knn_model.kneighbors(X_train_sample)
```

```
print("Test Point:\n", X_train_scaled.iloc[3])
print("Nearest Neighbors' Indices:", indices)
print("Nearest Neighbors' Distances:", distances)

# Show the features and labels of the nearest neighbors
for i, index in enumerate(indices[0]):
    #print(X_train[int(index)])
    print(f"Neighbor {i+1}\n, Index:\n {index}, Features:\n
{X_train_scaled.iloc[index]}, Outcome Label: {y_train.iloc[index]}")
```