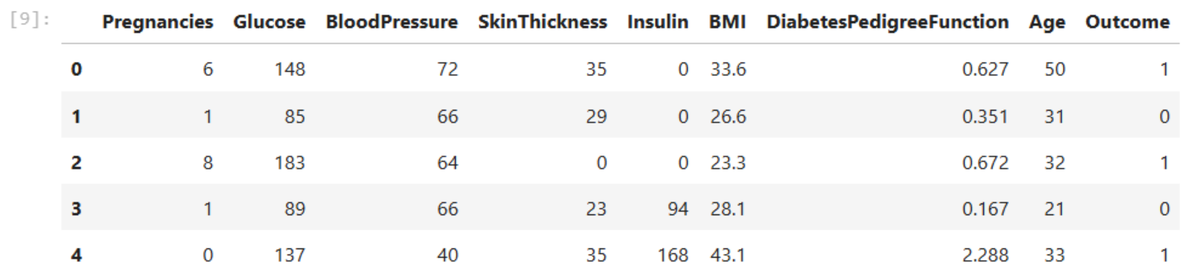
Predict Diabetes with Machine Learning

Abhinav Tiwari

According to the report of the Centers for Disease Control and Prevention, about one in seven adults in the United States has Diabetes. But over the next few years, this rate can move higher. With this in mind, today, in this article, I will show you how you can use machine learning to Predict Diabetes using Python.

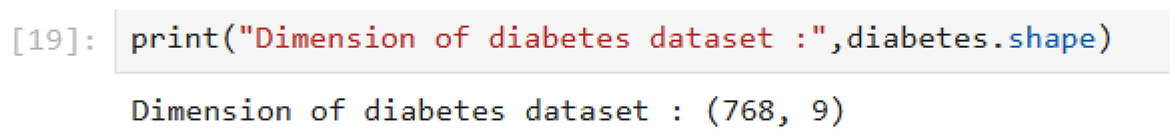
This is how a few entries of my Dataset look like :



[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

The diabetes data set consists of 768 data points, with 9 features each:



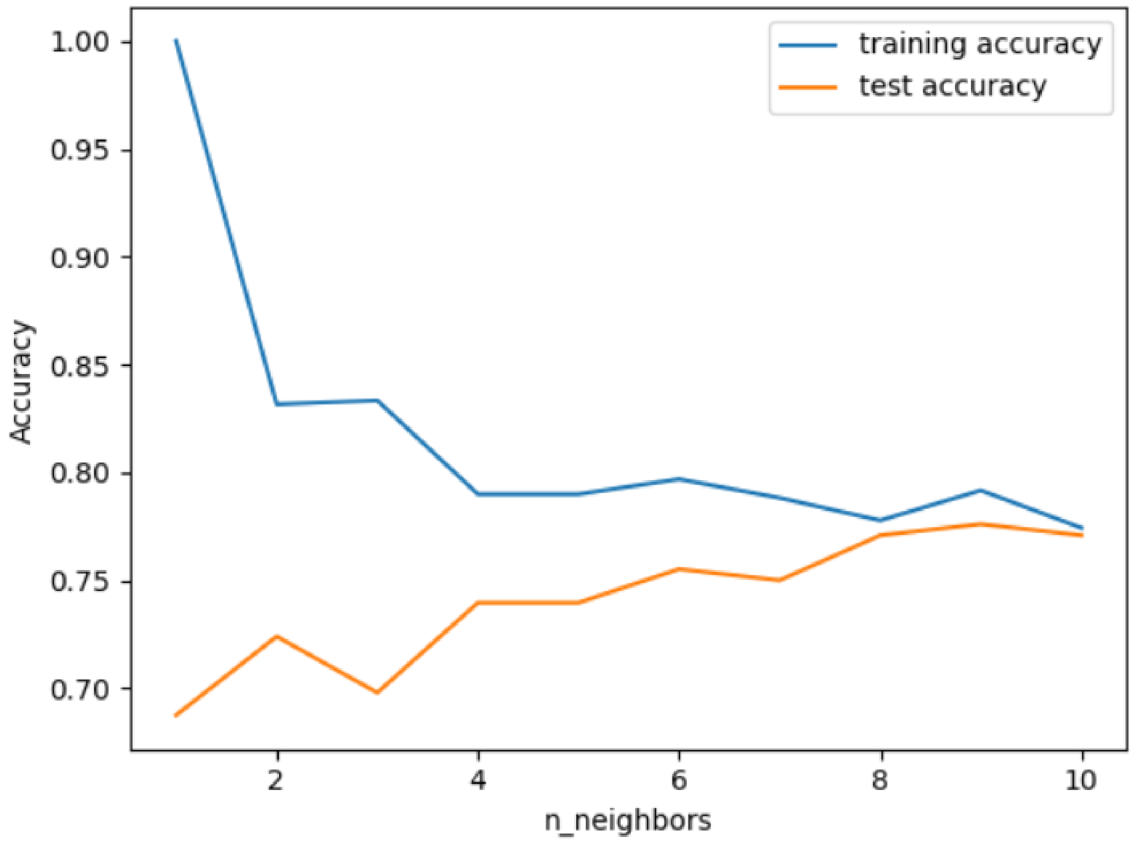
```
[19]: print("Dimension of diabetes dataset :",diabetes.shape)

      Dimension of diabetes dataset : (768, 9)
```

"Outcome" is the feature we are going to predict; 0 means no diabetes, 1 means diabetes. Of these 768 data points, 500 are labeled as 0 and 268 as 1:

K-Nearest Neighbours to Predict Diabetes

The k-Nearest Neighbours algorithm is arguably the simplest machine learning algorithm. Building the model consists only of storing the training dataset. To make a prediction for a new point in the dataset, the algorithm finds the closest data points in the training dataset — its “nearest neighbours.”

First, Let’s investigate whether we can confirm the connection between model complexity and accuracy:



Let’s check the accuracy score of the k-nearest neighbours algorithm to predict diabetes.

```
[41]: knn = KNeighborsClassifier(n_neighbors=9)
      knn.fit(X_train, y_train)
      print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(X_train, y_train)))
      print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_test, y_test)))
```

```
Accuracy of K-NN classifier on training set: 0.79
Accuracy of K-NN classifier on test set: 0.78
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.714
```

The accuracy on the training set with the Decision Tree Classifier is 100%, while the test set accuracy is much worse. This is an indication that the tree is overfitting and not generalizing well to new data. Therefore, we need to apply pre-pruning to the tree.

Now, I will do this again by setting max_depth=3, which limits the depth of the tree, decreasing overfitting. This leads to a lower accuracy on the training set, but an improvement on the test set.

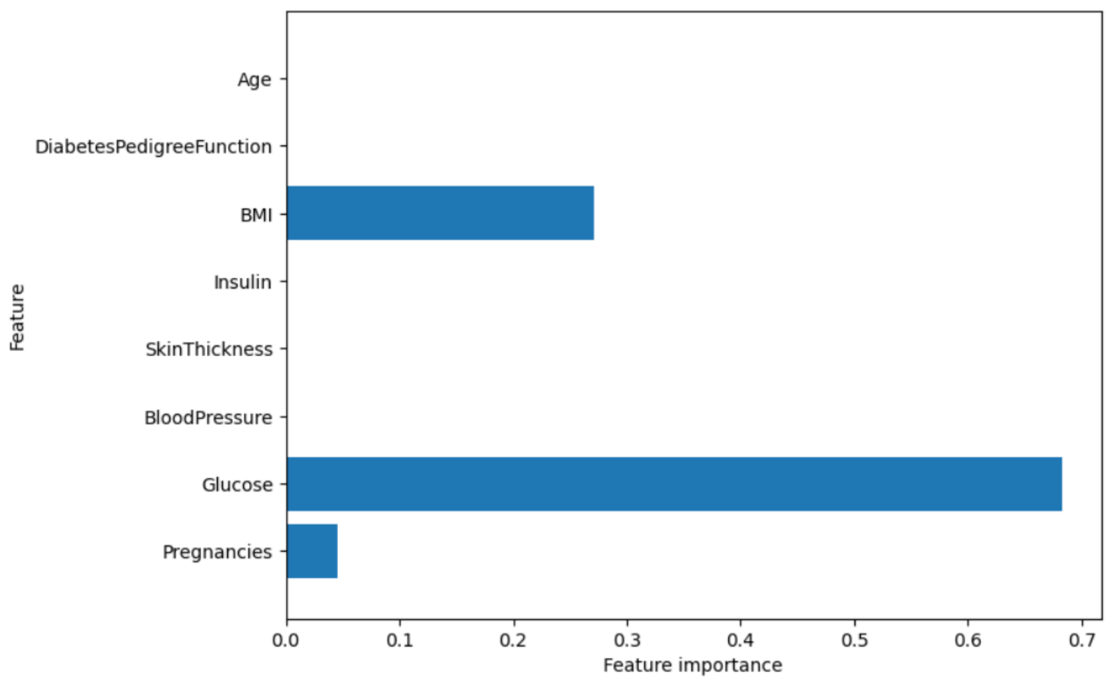
```
tree = DecisionTreeClassifier(max_depth=3, random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 0.773
Accuracy on test set: 0.740
```

Feature Importance in Decision Trees

Feature importance shows how important each feature is for the decision a decision tree classifier makes. It is a number between 0 and 1 for each feature, where 0 means “not used at all” and 1 means “perfectly predicts the target”. The feature importance always sum to 1.

Now lets visualize the feature importance of decision tree to predict diabetes.



So the Glucose feature is used the most to predict diabetes.

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(random_state=42)
mlp.fit(X_train, y_train)
print("Accuracy on training set: {:.2f}".format(mlp.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(mlp.score(X_test, y_test)))
```

```
Accuracy on training set: 0.73
Accuracy on test set: 0.72
```

The accuracy of the Multilayer perceptrons (MLP) is not as good as the other models at all, this is likely due to scaling of the data. Deep learning algorithms also expect all input features to vary in a similar way, and ideally to have a mean of 0, and a variance of 1. Now I will re-scale our data so that it fulfills these requirements to predict diabetes with a good accuracy.

Now let’s increase the number of iterations, alpha parameter and add stronger parameters to the weights of the model:

```
Accuracy on training set: 0.823
Accuracy on test set: 0.802
```

The result is good, but we are not able to increase the test accuracy further. Therefore, our best model so far is default deep learning model after scaling. Now I will plot a heat map of the first layer weights in a neural network learned on the to predict diabetes using the data set.