Machine learning Project - 17390281

Data Set: Pima Indians Diabetes Dataset

This dataset was chose from Kaggle for it's gold star rating and because the idea of applying machine learning techniques to medical science interests me a lot. This dataset comes from the **National Institute of Diabetes** (US) and provides a list of samples with different health factors and finally, whether or not the patient has diabetes.
The Pima people are a ethnic group of Native Americans living in Arizona. For reasons of location and diet, **they have the highest rates of type 2 diabetes in the world** which has led to many studies (Keen, H. and Jarrett, R.J. (1971); Baier, L.J. and Hanson, R.L. (2004); etc).

All patients are females at least 21 years old of Pima heritage.

Goal: To predict if a patient has diabetes based on the factors given

Step 1: The first step will be to import the CSV file and list all of the columns, determining what they are.

In[1]:=
```
rawdata = Import["C:\\Users\\anton\\Downloads\\diabetes.csv", "CSV"];
columns = rawdata[[1, All]];
keys = columns;

data = Dataset[AssociationThread /@ (Rule[keys, #] & /@ rawdata[[2 ;; 769]])];

Keys[data[[1]]]
```

Out[5]=

Pregnancies	Glucose
SkinThickness	Insulin
DiabetesPedigreeFunction	Age

Turning the raw data into a form I can use for later for classification

In[ ]:=
```
groupedData = GroupBy[rawdata[[2 ;; All]], Last → Most];
```

As we can see there are 9 columns in this CSV file. The first 8 reflect important patient metrics such as their age, blood pressure and number of pregnancies. Other metrics include glucose levels, skin thickness and BMI. The final column, Outcome, is a binary value which says whether or not the patient has diabetes.

Step 2: Scanning the data quickly to see what it looks like

In[ ]:=
```
data[[1 ;; 10]]
Length[data]
```

Out[ ]=

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
6	148	72	35	0	33.6	0.627
1	85	66	29	0	26.6	0.351
8	183	64	0	0	23.3	0.672
1	89	66	23	94	28.1	0.167
0	137	40	35	168	43.1	2.288
5	116	74	0	0	25.6	0.201
3	78	50	32	88	31	0.248
10	115	0	0	0	35.3	0.134
2	197	70	45	543	30.5	0.158
8	125	96	0	0	0	0.232

Out[ ]= 768

Looking at the data, it seems that there are some strange occurrences. For instance a value of 10 pregnancies seems a bit unusual and we would want to know what it means. After a bit of digging, I found a survey done by the Arizona Department of Health Services which included a comparison between the fertility rate of the *Salt River Pima-Maricopa Indian Community* with the average in Arizona. At 96.9 births per 1000 females compared to 61 births/1000 females for all of Arizona, there was a ~60% increase in the fertility rate of the Pima community compared to the Arizona average which gave some credence to the high pregnancy values the dataset was displaying.
(Source)

Another noticeable issue is in the *Insulin* column, there are values with only 0's which doesn't make any sense. Looking at Kaggle discussion boards, the explanation was simply that these are missing values. One possibility would be to impute this with either the median or the mean of the Insulin column. In order to see how useful this would be we take a look at the mean and standard deviation to see if the data is highly spread out or not.

In[ ]:=
```
cleandata = data[Select[#Insulin ≠ 0 &]];
Length[cleandata]
N[StandardDeviation[cleandata[[1 ;; 769, 5]]], 10]
N[Mean[cleandata[[1 ;; 769, 5]]], 10]
```

Out[ ]= 394

Out[ ]= 118.7758552

Out[ ]= 155.5482234

Since the data is so spread out, it would be unwise to try to impute the missing values with the mean. Since more than half the rows in the dataset have empty values in this column, losing the rows would If *Insulin* is stongly correlated with another column we can assume that keeping the other column wil

In[ ]:=
```
Preg = Normal[cleandata[[All, 1]]];
Gluc = Normal[cleandata[[All, 2]]];
Blo = Normal[cleandata[[All, 3]]];
Skin = Normal[cleandata[[All, 4]]];
Ins = Normal[cleandata[[All, 5]]];
Bmi = Normal[cleandata[[All, 6]]];
Dia = Normal[cleandata[[All, 7]]];
Age = Normal[cleandata[[All, 8]]];
N[Correlation[Preg, Ins], 5]
N[Correlation[Gluc, Ins], 5]
N[Correlation[Blo, Ins], 5]
N[Correlation[Skin, Ins], 5]
N[Correlation[Bmi, Ins], 5]
N[Correlation[Dia, Ins], 5]
N[Correlation[Age, Ins], 5]
```

```
graph1 = Transpose@{Ins, Gluc};
ListPlot[graph1, Frame → True, FrameLabel → {"Insulin", "Glucose"}]
```

Out[ ]= 0.082171

Out[ ]= 0.58001

Out[ ]= 0.098272

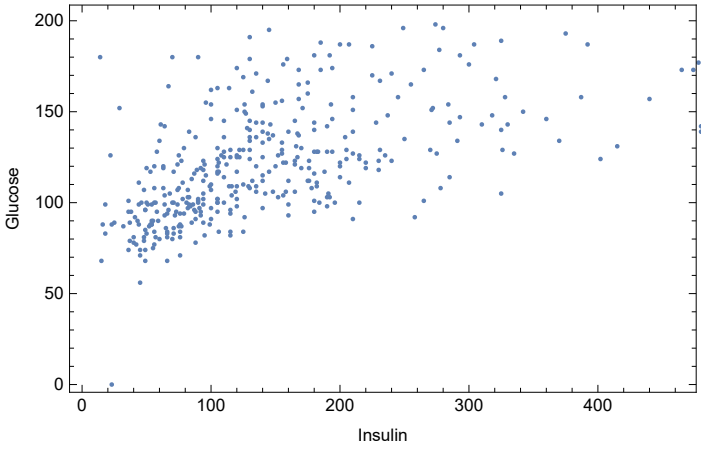
Out[ ]= 0.18489

Out[ ]= 0.228328

Out[ ]= 0.130395

Out[ ]= 0.22026

Out[ ]=



Looking at the outputs we can see that none of the other columns really have a strong correlation to Insulin, the closest being glucose levels, which inherently makes sense as glucose and insulin

production are intricately connected. Still at 0.5, the correlation is not great. The graph above demonstrates this quite clearly.

At this point I will work with the Insulin column included, but the rows with empty values excluded. Later on, I will try to remove the Insulin column instead and see if that helps the classification accuracy.

Step 3: Performing PCA to reduce the dimensions of the data

I will use both singular value decomposition and principal component analysis in order to reduce the dimensions of the data set to something which can hopefully perform just as well, and perhaps even be visualized.
I'll first convert the dataset back into a matrix, since Mathematica's dataset object doesn't work with PCA or SVD.

In[ ]:=
```
Export["cleandata.csv", cleandata];
A = Import["cleandata.csv", "CSV"];
A = A[[2 ;; 395, All]];
MatrixForm[A[[1 ;; 5]]]
```

Out[ ]//MatrixForm=

1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
3	78	50	32	88	31	0.248	26	1
2	197	70	45	543	30.5	0.158	53	1
1	189	60	23	846	30.1	0.398	59	1

Now that we have the clean data converted into a matrix, we want to standardize it by subtracting the mean of each column. We'll use the Standardize function to do this;

In[ ]:=
```
standardData = Standardize[A, Mean, 1 &];

MatrixForm[standardData[[1 ;; 5]]]
```

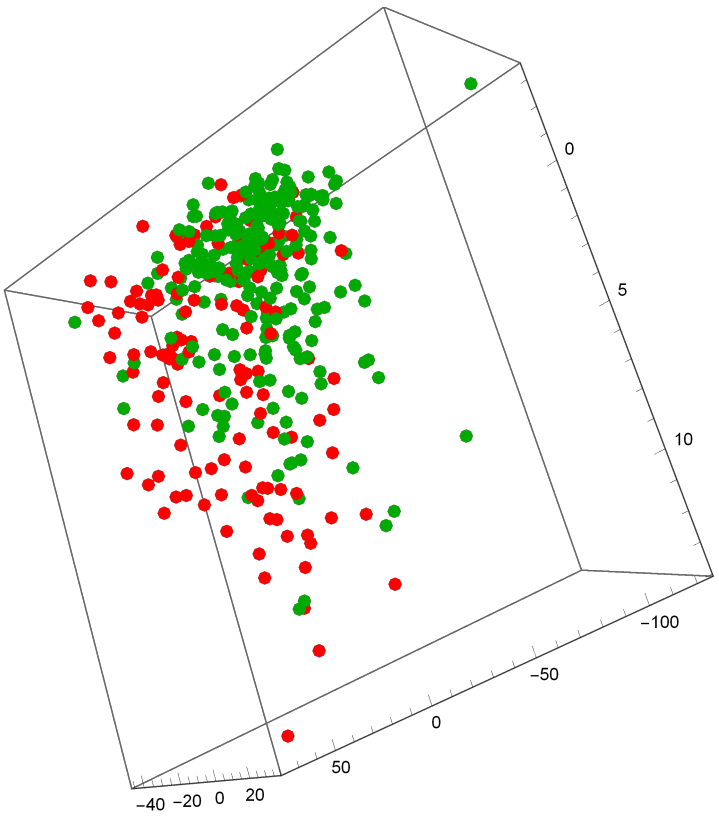
Out[ ]//MatrixForm=

−2.2868	−33.3046	−4.65482	−6.1066	−61.5482	−4.88858	−0.358543	−9.81472	−0.3299
−3.2868	14.6954	−30.6548	5.8934	12.4518	10.1114	1.76246	2.18528	0.67005
−0.286802	−44.3046	−20.6548	2.8934	−67.5482	−1.98858	−0.277543	−4.81472	0.67005
−1.2868	74.6954	−0.654822	15.8934	387.452	−2.48858	−0.367543	22.1853	0.67005
−2.2868	66.6954	−10.6548	−6.1066	690.452	−2.88858	−0.127543	28.1853	0.67005

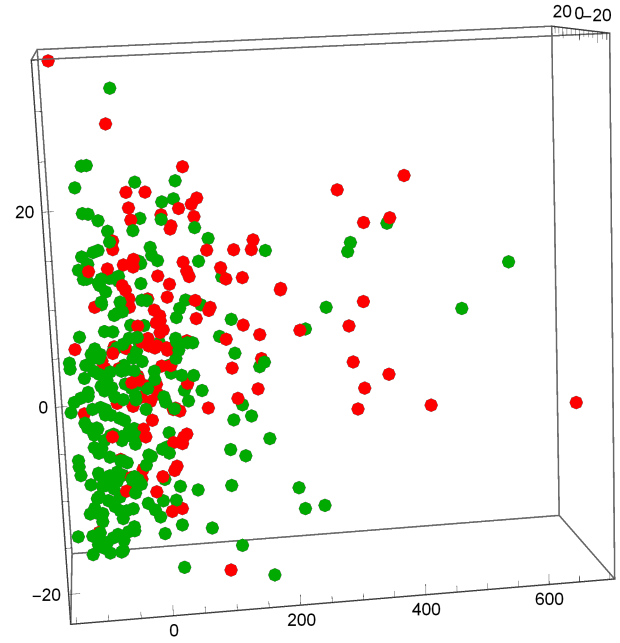
Lets group the data by the Output so that when we visualize it we can tell which data points correspond to diabetes/no diabetes. Since the best we can do is view things in 3D, I'll plot Age, BMI and Insulin as the spacial axis and use color-coding for the Outcome. This'll quickly tell us if we have a chance of using SVM to categorise things.
Here
Green -> No Diabetes
Red -> Diabetes.

In[•]:=
```
ListPointPlot3D[List /@ standardData[[All, {1, 2, 3}]],
 PlotStyle → ({PointSize[Large], Blend[{{-0.329, Darker[Green]}, {0.67, Red}}, #1]} & /@
    Flatten[standardData[[All, {9}]]])]
ListPointPlot3D[List /@ standardData[[All, {4, 5, 6}]],
 PlotStyle → ({PointSize[Large], Blend[{{-0.329, Darker[Green]}, {0.67, Red}}, #1]} & /@
    Flatten[standardData[[All, {9}]]])]
ListPointPlot3D[List /@ standardData[[All, {6, 7, 8}]],
 PlotStyle → ({PointSize[Large], Blend[{{-0.329, Darker[Green]}, {0.67, Red}}, #1]} & /@
    Flatten[standardData[[All, {9}]]])]
```

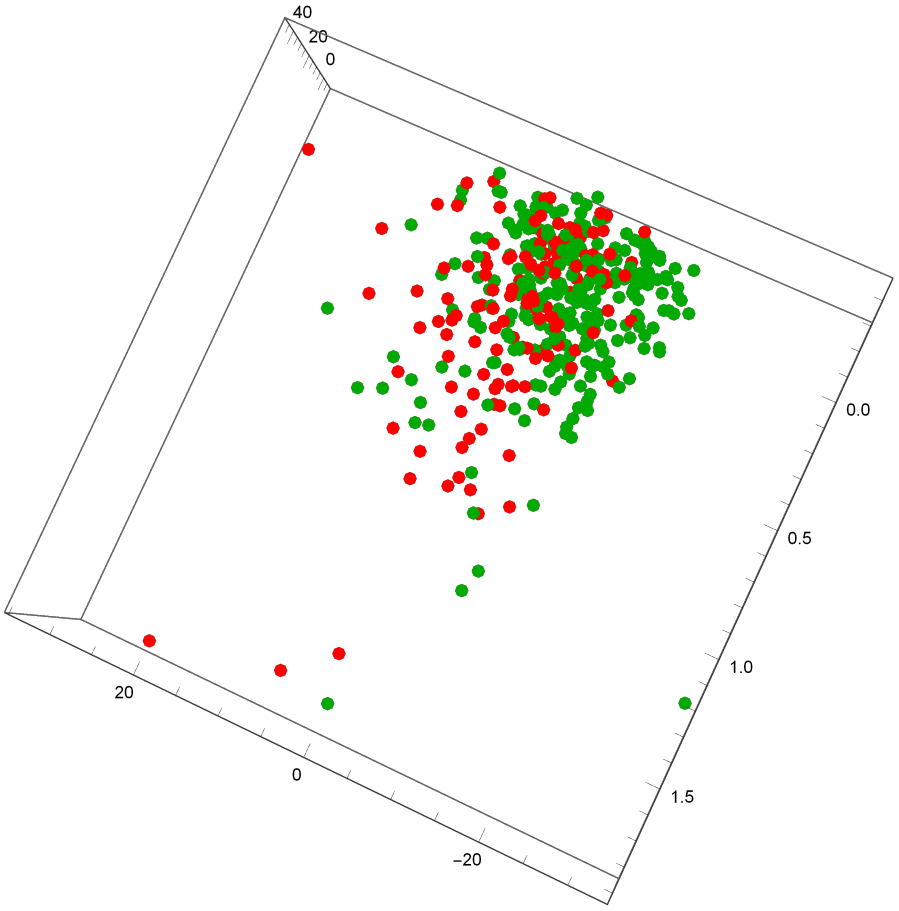
Out[•]=



Out[ ]=



Out[ ]=



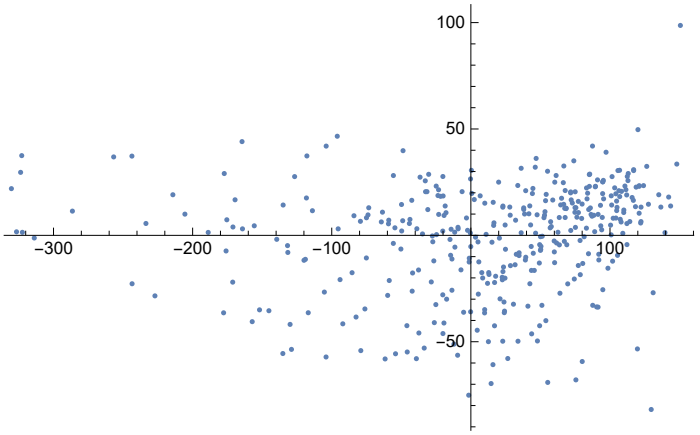
Looking at the 3D plots we can see that there is some separation between the diabetic patients and the non-diabetic ones although for most of them it is not fully distinct how to separate them.

Now we use Singular Value Decomposition to get the principal components then then project the data along the principal components to see what it looks like. We'll colour code it again to make sure we can tell the output's apart from each other.

In[ ]:=
```
groupedStandard = GroupBy[standardData, Last → Most];
```

In[ ]:=
```
{U, Σ, V} = SingularValueDecomposition[standardData[[All, 1 ;; 8]]];
σs = Diagonal[Σ];
ListPlot[U[[All, 1 ;; 2]].Σ[[1 ;; 2, 1 ;; 2]]]
```

Out[ ]=



To keep the colour scheme I'll use the DimensionReduction function instead.

In[ ]:=
```
reducer = DimensionReduction[
  standardData[[All, 1 ;; 8]], 2, Method → "PrincipalComponentsAnalysis"]
reducer3d = DimensionReduction[standardData[[All, 1 ;; 8]],
  3, Method → "PrincipalComponentsAnalysis"]
```

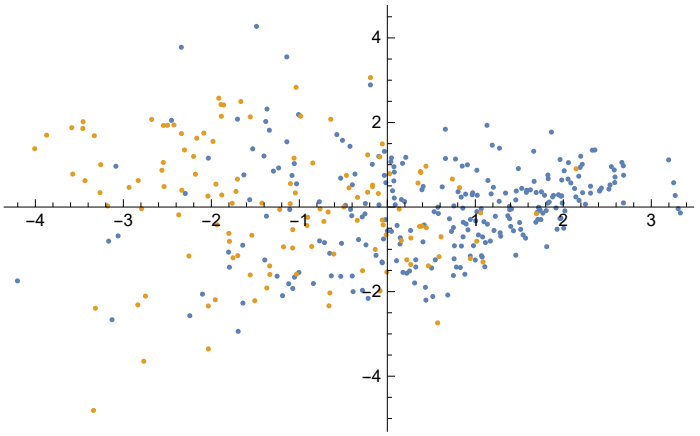
Out[ ]= DimensionReducerFunction[Input type: NumericalVector (length: 8) Output dimension: 2]

Out[ ]= DimensionReducerFunction[Input type: NumericalVector (length: 8) Output dimension: 3]

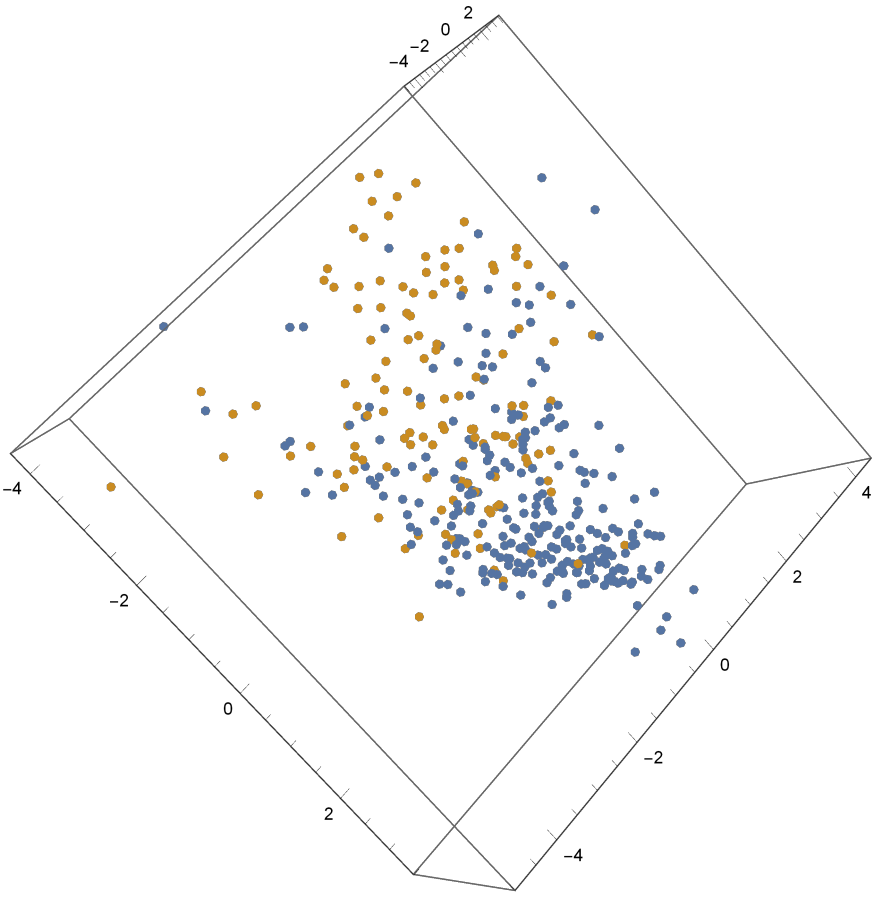
In[ ]:=
```
reducedvalues = reducer /@ groupedStandard;
reducedvalues2 = reducer3d /@ groupedStandard;

ListPlot[{reducedvalues[[1]], reducedvalues[[2]]}]
ListPointPlot3D[reducedvalues2]
```

Out[ ]=



Out[ ]=



Step 4: Classifying the Data

We can see more clearly here that there is a distinction that appears between the Diabetic and Non Diabetic patients, which gives us the confirmation that we can use Support Vectors to classify it. We'll now divide the data into three parts; Testing data, Validation data and Training data. I will feed the Training & Validation data to Mathematica's *Classifier* function and let it build predictive accuracy. Then I will try it out on the Testing data to see how well it performs.

```
In[ ]:= rs1 = RandomSample[Range[Length[A]]];
        train1 = rs1[[1 ;; Round[Length[A] * 0.6]]];
        validation1 = rs1[[Round[Length[A] * 0.6] + 1 ;; Round[Length[A] * 0.8]]];
        test1 = rs1[[Round[Length[A] * .8] + 1 ;;]];
```

```
In[ ]:= groupedTrain1 = GroupBy[A[[train1]], Last → Most];
        groupedVal1 = GroupBy[A[[validation1]], Last → Most];
        groupedTest1 = GroupBy[A[[test1]], Last → Most];

        Length[groupedTrain1[[1]]] + Length[groupedTrain1[[2]]]
        Length[groupedVal1[[1]]] + Length[groupedVal1[[2]]]
        Length[groupedTest1[[1]]] + Length[groupedTest1[[2]]]
```

```
Out[ ]= 236
```

```
Out[ ]= 79
```

```
Out[ ]= 79
```

```
In[ ]:= c = Classify[groupedTrain1,
          ValidationSet → groupedVal1, Method → "SupportVectorMachine"]
```

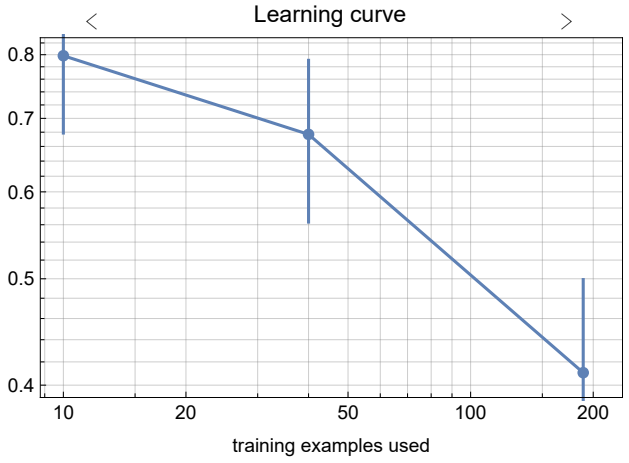
```
Out[ ]= ClassifierFunction[ Input type: Mixed (number: 8)
                            Classes: 0, 1 ]
```

In[ ]:=
```
Information[c]
```

Out[ ]=

Classifier information

Data type	Mixed (number: 8)
Classes	0, 1
Accuracy	(82. ± 6.)%
Method	SupportVectorMachine
Single evaluation time	5.23 ms/example
Batch evaluation speed	22.4 examples/ms
Loss	0.411 ± 0.089
Model memory	253. kB
Training examples used	236 examples
Training time	3.45 s



The Mathematica Classifier Function will use the dual problem method as well as the Lagrange multipliers to build a decision line (or in this case a hyperplane) which divides up the data into two sets, the diabetics and non-diabetics. While the accuracy of the model is currently at 83%, in order to see a more realistic accuracy, we'll try the model on the testing data.

In[ ]:=
```
testDiabetic = c /@ groupedTest1[[1]]
testNondiabetic = c /@ groupedTest1[[2]]
```

Out[ ]=
```
{0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

Out[ ]=
```
{0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0}
```

Here the first output above, corresponds to Test data from only non-diabetics, which means we expect a value of 0 as the output. To get the accuracy of the testing data, I will tally up the number of times the wrong output is given for both 1's and 0's.

In[ ]:=
```
Tally[testDiabetic]
Tally[testNondiabetic]
```

Out[ ]=
```
{{0, 47}, {1, 7}}
```

Out[ ]=
```
{{0, 8}, {1, 17}}
```

Here the model is 86% accurate for detecting non-diabetics and 66% accurate for detecting diabetics using the testing data which seems good. However, we have only 79 values in the testing data which is very low and should raise some questions about how legitimate the accuracy rating is.

Step 5: Adjusting the data to optimize the model

I'm going to try to use the Classifier on the data after PCA has reduced it from 8 dimensions to 2 dimensions, and then see if this helps with the accuracy of the model. I'll keep the data which has the zero insulin values removed from it for consistency.

In[•]:=
```
reducedtrain1 = reducer /@ GroupBy[standardData[[train1]], Last → Most];
reducedval1 = reducer /@ GroupBy[standardData[[validation1]], Last → Most];
reducedtest1 = reducer /@ GroupBy[standardData[[test1]], Last → Most];
```

In[•]:=
```
c1 = Classify[reducedtrain1,
  ValidationSet → reducedval1, Method → "SupportVectorMachine"]
```

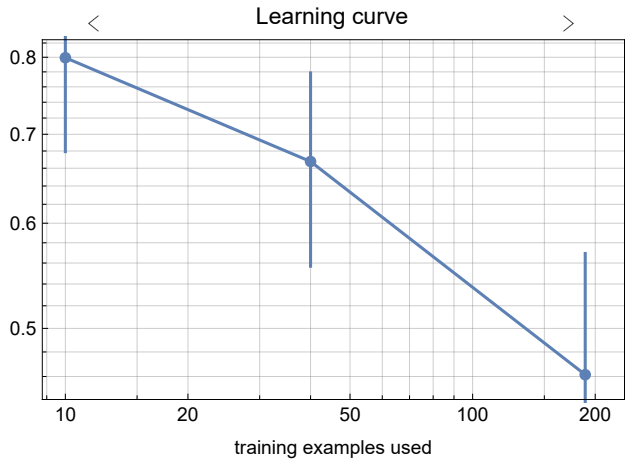
Out[•]= ClassifierFunction[ Input type: NumericalVector (length: 2)
Classes: −0.33, 0.67]

In[ ]:=
```
Information[c1]
```

Out[ ]=

Classifier information

Data type	NumericalVector (length: 2)
Classes	−0.329949, 0.670051
Accuracy	(78. ± 7.)%
Method	SupportVectorMachine
Single evaluation time	3.15 ms/example
Batch evaluation speed	58.4 examples/ms
Loss	0.461 ± 0.11
Model memory	215. kB
Training examples used	236 examples
Training time	2.25 s



Looking at the information data, the accuracy of the new classifier is a bit shorter than the previous one. However considering the memory boost, dropping 6 columns of data would provide, this is pretty good. Of course, in order to get a much better idea of the performance of this new model, we'll tally up the results with the testing data and see how it performs now.

In[ ]:=
```
testDiabetic1 = c1 /@ reducedtest1[[1]]
testNondiabetic1 = c1 /@ reducedtest1[[2]]
```

Out[ ]=
```
{-0.329949, -0.329949, -0.329949, -0.329949, -0.329949, 0.670051,
 -0.329949, -0.329949, 0.670051, 0.670051, -0.329949, -0.329949, -0.329949,
 0.670051, 0.670051, -0.329949, -0.329949, -0.329949, 0.670051, -0.329949,
 -0.329949, -0.329949, -0.329949, -0.329949, -0.329949, -0.329949,
 -0.329949, -0.329949, -0.329949, -0.329949, -0.329949, -0.329949, -0.329949,
 -0.329949, -0.329949, -0.329949, -0.329949, -0.329949, -0.329949, -0.329949,
 0.670051, -0.329949, -0.329949, -0.329949, -0.329949, -0.329949, -0.329949,
 -0.329949, -0.329949, 0.670051, -0.329949, -0.329949, -0.329949, -0.329949}
```

Out[ ]=
```
{-0.329949, 0.670051, -0.329949, 0.670051, 0.670051, -0.329949,
 0.670051, 0.670051, -0.329949, -0.329949, 0.670051, -0.329949,
 0.670051, 0.670051, -0.329949, 0.670051, 0.670051, 0.670051, -0.329949,
 -0.329949, 0.670051, -0.329949, -0.329949, 0.670051, -0.329949}
```

```
In[•]:= Tally[testDiabetic1]
        Tally[testNondiabetic1]
Out[•]= {{-0.329949, 46}, {0.670051, 8}}
Out[•]= {{-0.329949, 12}, {0.670051, 13}}
```

Here we see the new model performed almost similarly as the old one, except it was one data point better, again, with the size of the testing data it's hard to verify if this is a genuine benefit but it appears that the PCA method has allowed us to create a second classifier which performs just as well as the first, except now it only takes in 2 dimensions, rather than 8.

Step 6: Removing the Insulin Column

Earlier (at Step 2) I decided to remove all the rows which had a null value for the Insulin column, as this would negatively affect the model. I will now, try the second possibility, that is, I will remove the entire *Insulin* column but keep the rows that remain. This should give much more data values to work with in the model which should hopefully help things.

```
In[•]:= B = Drop[rawdata, None, {5}] ;
        B = B[[2 ;; 769, All]];
```

Since I've already used *PCA* and *ListPointPlot3D* to see how distinct both categories are, I will simply proceed directly to the Classifier.

```
In[•]:= rs2 = RandomSample[Range[Length[B]]];
        train2 = rs2[[1 ;; Round[Length[B] * 0.6]]];
        validation2 = rs2[[Round[Length[B] * 0.6] + 1 ;; Round[Length[B] * 0.8]]];
        test2 = rs2[[Round[Length[B] * .8] + 1 ;;]];

In[•]:= groupedTrain2 = GroupBy[B[[train2]], Last → Most];
        groupedVal2 = GroupBy[B[[validation2]], Last → Most];
        groupedTest2 = GroupBy[B[[test2]], Last → Most];

        Length[groupedTrain2[[1]]] + Length[groupedTrain2[[2]]]
        Length[groupedVal2[[1]]] + Length[groupedVal2[[2]]]
        Length[groupedTest2[[1]]] + Length[groupedTest2[[2]]]
Out[•]= 461
Out[•]= 153
Out[•]= 154
```

We now have 461 data points for training compared to the 236 points we had previously along with 154 values for the testing data. Ideally, this should lead to a significant boost in the predictive power of the model. However, this comes at the assumption that the '*Insulin*' column was not providing the majority of the predictive power in the last classifier.
I will create the Classifier and see how it performs now;

In[ ]:=
```
c2 = Classify[groupedTrain2,
  ValidationSet → groupedVal2, Method → "SupportVectorMachine"]
```

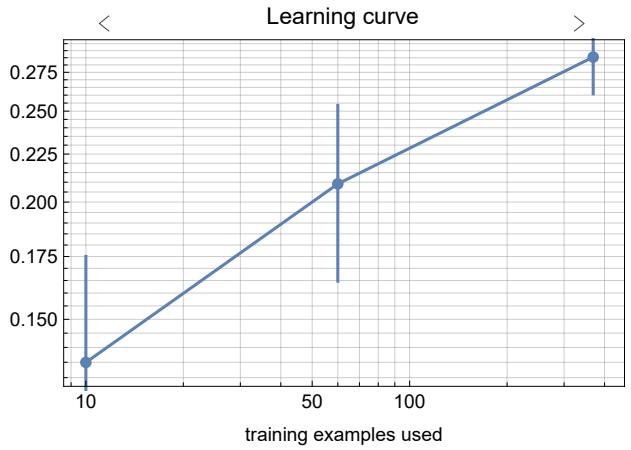
Out[ ]=
```
ClassifierFunction[ Input type: Mixed (number: 7)
                    Classes: 0, 1 ]
```

In[ ]:=
```
Information[c2]
```

Out[ ]=

Classifier information

Data type	Mixed (number: 7)
Classes	0, 1
Accuracy	(88.5 ± 3.4)%
Method	SupportVectorMachine
Single evaluation time	4.72 ms/example
Batch evaluation speed	21.5 examples/ms
Loss	0.135 ± 0.040
Model memory	282. kB
Training examples used	461 examples
Training time	3.7 s



A training accuracy of 90% seems great, and is an improvement over the previous two training models which had an accuracy of 83%. Like before, to obtain a more realistic accuracy, we'll apply the classifier to the testing data and see how it goes;

In[ ]:=
```
testDiabetic2 = c2 /@ groupedTest2[[1]]
testNondiabetic2 = c2 /@ groupedTest2[[2]]
```

Out[ ]=
```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0}
```

Out[ ]=
```
{1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1,
 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0}
```

```
In[•]:= Tally[testDiabetic2]
        Tally[testNondiabetic2]
```

```
Out[•]= {{0, 83}, {1, 15}}
```

```
Out[•]= {{1, 30}, {0, 26}}
```

Looking at the results, the new classifier has an accuracy of 54% for classifying the diabetics, and 86% for classifying non diabetics. So what's going on here? Well with the majority of the dataset containing non-diabetics, the model is biased towards labeling patients as non-diabetic. By doing so, it's overall accuracy can still be high, even if it's highly inaccurate at classifying diabetic patients. In the previous model, there was a more even split between diabetics and non-diabetics which lead to better accuracy for diabetics, but a lower training accuracy.

How do we fix this and keep a large dataset? Matthew Stewart has an article where he goes over potential remedies for this problem. The most common and easy solution would be to balance the model out. This means ideally a 50-50 split between diabetics and non-diabetics in our training and validation data so that the *Classifier* is forced to notice their subtleties a bit more. First I'll check just how the values are split between diabetics and nondiabetics in the current training & validation data sets.

```
In[•]:= {Length[groupedTrain2[[1]]], Length[groupedTrain2[[2]]]}
        {Length[groupedVal2[[1]]], Length[groupedVal2[[2]]]}
        {Length[groupedTest2[[1]]], Length[groupedTest2[[2]]]}
        Length[A]
```

```
Out[•]= {308, 153}
```

```
Out[•]= {94, 59}
```

```
Out[•]= {98, 56}
```

```
Out[•]= 394
```

We can see there's over 100 more values for non-diabetics in the training data and over twice as many in the validation data. If I shave off 100 rows for the training data and 50 rows for the validation data, I can get a much more even split. Theoretically, that should help the classifier.

```
In[•]:= trimmedtrain = groupedTrain2[[1]][[1 ;; 175]];
        trimmedval = groupedVal2[[1]][[1 ;; 45]];
```

```
In[•]:= newgroupedTrain = <|0 → trimmedtrain, 1 → groupedTrain2[[2]]|>;
        newgroupedVal = <|0 → trimmedval, 1 → groupedVal2[[2]]|>;
```

```
In[•]:= c3 = Classify[newgroupedTrain,
          ValidationSet → newgroupedVal, Method → "SupportVectorMachine"]
```

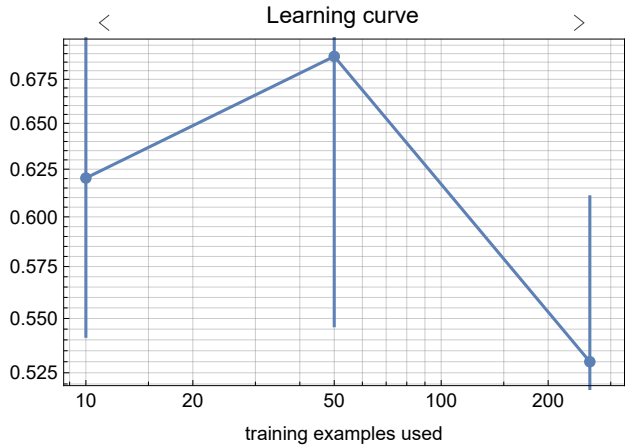
```
Out[•]= ClassifierFunction[ Input type: Mixed (number: 7)
                            Classes: 0, 1 ]
```

In[ ]:=
```
Information[c3]
```

Out[ ]=

Classifier information

Data type	Mixed (number: 7)
Classes	0, 1
Accuracy	(76. ± 6.)%
Method	SupportVectorMachine
Single evaluation time	3.31 ms/example
Batch evaluation speed	55.2 examples/ms
Loss	0.530 ± 0.080
Model memory	279. kB
Training examples used	328 examples
Training time	1.5 s



The new model has a much lower percentage of accuracy at 76%. Hopefully, it will do better at recognizing diabetic patients with the testing data.

In[ ]:=
```
testDiabetic3 = c3 /@ groupedTest2[[1]]
testNondiabetic3 = c3 /@ groupedTest2[[2]]
```

Out[ ]=
```
{1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0,
 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0}
```

Out[ ]=
```
{1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1}
```

In[ ]:=
```
Tally[testDiabetic3]
Tally[testNondiabetic3]
```

Out[ ]=
```
{{1, 42}, {0, 56}}
```

Out[ ]=
```
{{1, 51}, {0, 5}}
```

We can immediately see that the Classifier has a huge improvement in the accuracy of the testing data! Now it's identified diabetic's correctly 91% of the time, compared to 54% in the C2 model. As for non-diabetics, the accuracy has dropped to 58% of the time, which is poorer than before. The most most important result here, is that now the model is less biased towards non-diabetics and

has a fairly high level of accuracy for predicting diabetes.

Conclusions

The goal of this project was to build a simple machine learning model which would use the *Pima Indians* dataset to analyze medical data from female patients and conclude if they had diabetes. In order to do this a few slightly different models were considered, all of which revolved around using Support Vector Machines to classify the data. When scanning the data, it was noticed that a column corresponding to *Insulin* values had a large amount of empty entries. Since this would introduce problems into the model, the data was cleaned by two different techniques. One was to remove the entire Insulin column, while the other was to remove all rows in the dataset which had no entries for that column. Both methods attempted, the former was found to perform slightly better. Using principal component analysis on the data gave rise to 2D and 3D graphs which showed the split between the diabetics and non-diabetics in the data. This served as a way to determine if support vectors would work or if a kernel trick was also required. In this case, no kernel trick was necessary. A testing accuracy of approx 91% was the best that was observed in the model and came from allowing a ~ 50:50 split in the training and validation data, so that the non-diabetics would not dominate the model. Prior to this, the accuracy of the model for predicting diabetics capped at around 76%. While this seems like a very good improvement, the model could be made much better by a bigger sample size to test on as well as more training/validation data. In the future, this would be the two main things to improve on. Finally, as Mathematica's *Classifier* function appeared to give different results each time it was used, there is a level of uncertainty as to when it performs better in one run vs another run.