Data Mining and Warehousing

Module leader: **Dr. Walid Hussein**

Group:3

https://colab.research.google.com/drive/1jpBuVPtc4MaNfBOSd07E3v-iFqlQZY7s?usp=sharing

Name	ID	Major
Belal Ahmed	191121	SE
Anas Abdulatif	189605	SE
Abdelrahman Zakaria	186883	SE

eBay shill bidding clustering

Introduction

In our quest to find the best classification algorithm to identify suspicious auction bids we will try to Support Victor Machine and neural network. Support Victor Machine purpose is to increase the dimensionality of data to fit a support victor classifier.

> A **Support Victor Classifier** uses a **soft margin** to fit the data. A **margin** in machine learning means the distance between the classifier and two points of the same distance from the classifier. That distance is the margin. But to understand a soft margin first we must understand **maximum margin**. To clarify how maximum margin classifier works, we will consider having two clusters. In a maximum margin we find the closest two points between the clusters and taking the maximum margin between the two points. In case there is no misclassification the maximum margin allows for the biggest size between the two clusters. This allows for high accuracy and shows the distinction between the two classes. But in case of misclassification, maximum margin accuracy will drastically decrease. This is because maximum margin is very sensitive to misclassification. That's why soft margin was created. Soft margin allows the classifier to ignore data points if it will increase accuracy. The number of datapoints ignored is usually determined by cross validation.

The reason why SVC isn't used alone is because it can't fit some data formats. For example, a one-dimensional data of medicine dosage where low amount of dosage doesn't cure the sick, the right amount cures while a big amount doesn't cure. A SVC can't fit this data set. Thus, we need to increase the dimensionality of data. We can do this by for example maxing a y axis that equals x axis squared. This is what SVM does. The part in SVM responsible for this the SVM kernel. But kernels don't increase he dimensionality of the data they just calculate the high dimensional relationship between datapoints without doing the transformation from low dimensionality to high dimensionality. There are multiple types of kernels that can be used and each of them calculates the high dimensional relationship between datapoints differently. The most well-known kernels are **polynomial** and **radial kernel.**

Polynomial kernel

Which calculates the relationship between datapoint in defined dimensions by using this formula

$$(a \times b + r)^d$$

Where a and b are data points locations

R the coefficient of the polynomial

D Is the degree of the polynomial

Given a polynomial with a coefficient of ½ and degree of 2

The resulting polynomial will be

$$ab + a^2b^2 + \frac{1}{4}$$

Which will equal this dot product which is what a kernel calculates

$$(a, a^2, \frac{1}{2}) \cdot (b, b^2, \frac{1}{2})$$

A is found by splitting the multiplications.

The above coordination is for 3 dimensions. But since the z-axis is equal in both coordination, we can ignore the last axis. This means the number of dimensions is equal to the degree of the polynomial.

Radial kernel

The radial kernel calculates the high dimensional relationship in infinite dimensions. The formula to calculate. Since infinite dimensions, it cannot be visualized. But the radial kernel behaves like a weighted nearest neighbor where the close datapoint classification has a huge influence on the classified datapoint classification.

$$e^{-\gamma(a-b)^2}$$

A-b is the distance between the two data points and squares to remove the negativity since distance cannot be negative.

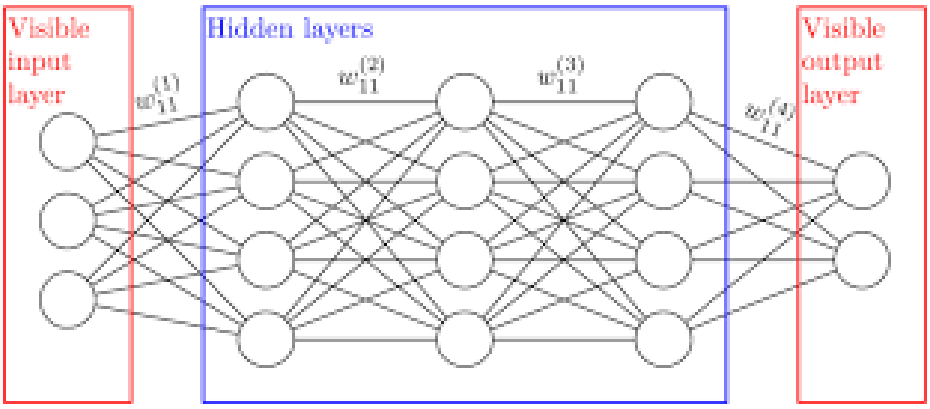
Gamma is for scaling the influence of distance and it is defined by cross-validation.

Radial kernel is said to be calculating the relationship in infinite dimensions because due to **Tylor series expansion**

$$f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\cdots+\frac{f^{\infty}(a)}{\infty!}(x-a)^{\infty}$$

which states that a function can be split into an infinite sum if infante derivatives exist. Since the derivative of e is e there exist infinite derivatives for e

neural networks are a different branch of AI from machine learning this due to drastic change in architecture from the normal machine learning model. A neural network consists of notes and connections between the nodes. A neural network consists of an input layer, hidden layer, and output layer.



The input layer contains nodes that take data of the datapoints and forward it to the connections that lead to the hidden layer

the connections between nodes contain weights and biases

weights are the values multiplied to the forwarded data

biases are values added to the forwarded data

both weights and biases are determined using backpropagation or the training of the neural network. Weights are biases are analogous to slope and intercept in linear regression.

Hidden layers contain nodes that contain activation functions that take values forwarded to them and calculate a y-axis location of the original input data point. Each hidden node draws a function on the main graph. Summing the hidden layer functions with consideration to the weights and bias gives the neural network classification function which is a low bias classification function.

The output layer produces the classification, or the regression value estimated by the neural network.

The backpropagation estimates the value for the weights and bias using the chain rule between the anonym parameter in relation to the sum of all residuals. To shorten the amount of guess for weights and biases gradient descent is used. The backpropagation is repeated until the sum of all residuals cannot be decreased anymore.

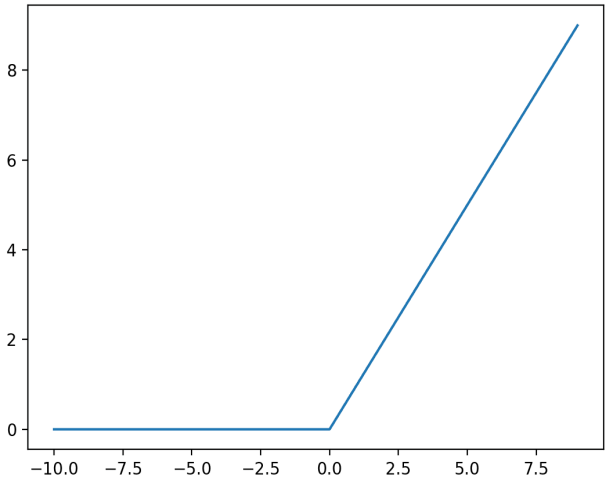
There are several activation functions used in neural networks below. One of the most used activation functions is.

Rectified Linear Unit (ReLU)

Its formula is

$$\max(0, x)$$

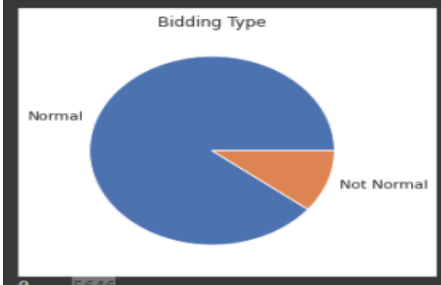
This means that reLU forward 0 if the forwarded value is smaller than 0 and forward the value as is if the value is bigger than 0. Thus, the activation function looks like this.



2- Prepossessing

There are 5646 records of normal record while there is 657 thus our dataset is unbalanced.

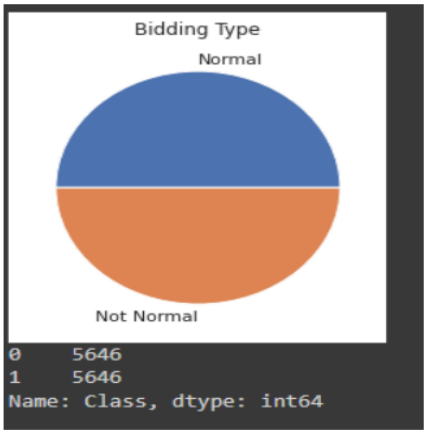
```
classes=df['Class'].value_counts()
plt.pie(classes,labels = ['Normal', 'Not Normal'])
plt.title ("Bidding Type")
plt.show()
classes
```




```
0    5646
1     675
Name: Class, dtype: int64
```

7

Thus, Oversampling or an under-sampling technique was required to balance our dataset. Since our minority class is rather small only oversampling techniques were viable. We tried both Random Over Sampler and smote. Smote results we superior to Random Over Sampler so we used it.



To create our classification algorithm, the ID of the auction and the bidder and the record is all irrelevant. Thus, the first pre-processing is dropping those features. The second pre-processing is confirming that there are no missing values. Third, pre-processing is dropping the duplicates. Our data contains no categorical attributes thus, no further pre-processing is required.

The results showed that the smote create overfitting in our model. thus, we had to try to train the model without smote even if the data is unbalanced.

ANN (With and Without SMOTE)

An artificial Neural Network is a computation model that works as the nerve cells function in the human brain. In the first step creating a class of Keras named by sequential which allows us to create models layer-by-layer for our problem.

```
ann = models.Sequential([
```

Then after that, we create the hidden layers which are not visible to the external systems by using the Dense class which is a part of the layers module. While the 'relu' will be the activation function that decides whether a neuron should be activated or not as 'relu' helps to prevent the exponential growth in the computation.

```
ann = models.Sequential([
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
```

And after that, we created the Output layer which is responsible for the result and it must have sigmoid as an activation function because it is a binary classification problem.

```
  layers.Dense(2, activation='sigmoid')
```

Then after creating the layers, we will compile our network and for the compile function we will need to use optimizers as we used "SGD", in addition to this we will need a loss function and then we will add the metrics which is used to calculate the performance and our performance metric is "accuracy"

```
ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

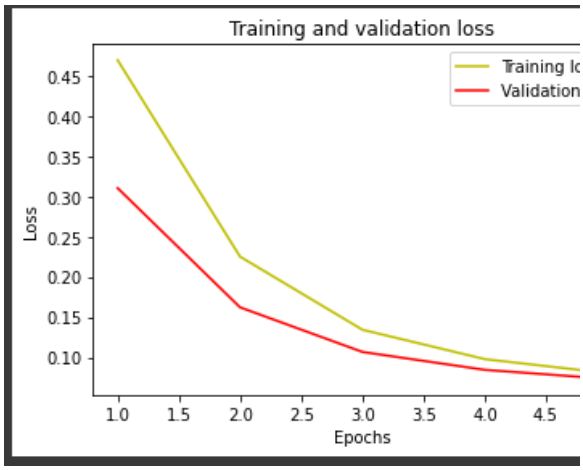
And the last step in the ann creation is to train the model on the training dataset by defining the X-train and Y-train that the X-train represents the features for the training dataset, while the Y_train is a dependent variable vector for the training dataset. And finally how many times the neural network will be trained is represented in the epochs count as we trained our model 5 times.

```
history = ann.fit(X_train, y_train ,verbose=1, epochss=5,
                  validation_data=(X_test, y_test))
```

ANN With SMOTE

```
from sklearn.metrics import confusion_matrix , classificat
yp = ann.predict(X_test)

y_pred = []

for element in yp:

    if element[0] > 0.5:
        y_pred.append(0)

    else:
        y_pred.append(1)


print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      2261
           1       0.98      1.00      0.99      2256

    accuracy                           0.99      4517
   macro avg       0.99      0.99      0.99      4517
weighted avg       0.99      0.99      0.99      4517
```

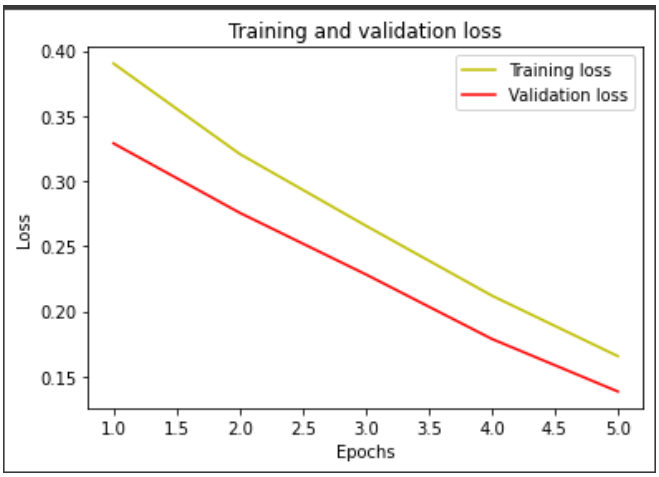


The training loss shows how well the model is fitting the training data while the validation loss indicates how well the model fits the new data after plotting it, in the first epoch both the validation loss and training loss were their maximum, and by increasing iteration both decreased which shows by increasing the epochs it affects the training and validating loss negatively which indicates the model is over-fitting.

ANN Without SMOTE

```
[86] from sklearn.metrics import confusion_matrix , classification_report
     yp = ann.predict(X_test2)

     y_pred = []

     for element in yp:

         if element[0] > 0.5:
             y_pred.append(0)

         else:
             y_pred.append(1)


     print(classification_report(y_test2,y_pred))
```

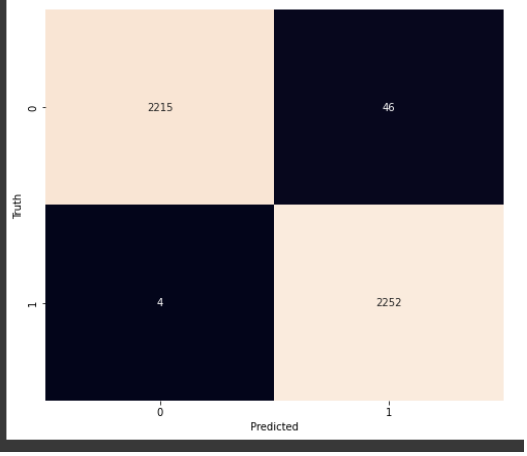
```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      2271
           1       0.93      0.69      0.79       258

    accuracy                           0.96      2529
   macro avg       0.95      0.84      0.88      2529
weighted avg       0.96      0.96      0.96      2529
```



By applying the same plotting of the training and validation loss the same deceleration appeared but it was better at the last epoch as it reached 0.15 while by applying SMOTE it reached 0.10 and the same goes for the training loss it was nearly to be 0.20 while in the SMOTE it reached the 0.10.

```
import seaborn as sn
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_pred)

plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(69.0, 0.5, 'Truth')
```



In this confusion matrix, the true positive (TP) equals 2215 while the false positive (FP) equals 46 and the false negative (FN) values equal 4, and finally, the true negative values (TN) equal 2252

Accuracy

```
ann.evaluate(X_test,y_test)
```

```
142/142 [==============================] - 1s 6ms/step - loss: 0.0728 - accuracy
[0.07276797294616699, 0.9873809814453125]
```

The accuracy reached by applying ANN with SMOTE 0.9874 which is 98.74%

After calculating some findings from the confusion matrix which identify the accuracy such as

ACC = (TP + TN) / (P + N) = 0.9889 = 98.89%

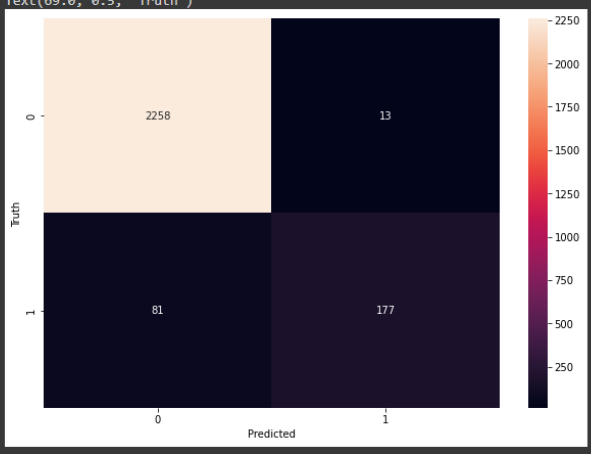
Precision = TP / (TP + FP) = 0.9797 = 97.97%

F1 score = 2TP / (2TP + FP + FN) = 0.9888 = 98.88%

Recall = TP / (TP + FN) = (2215) / (2215 +4) = 0.998 = 99.8

```
import seaborn as sn
cm = tf.math.confusion_matrix(labels=y_test2,predictions=y_pred)

plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(69.0, 0.5, 'Truth')
```



In this confusion matrix, the true positive (TP) equals 2258 while the false positive (FP) equals 13 and the false negative values equal 81, and finally, the true negative values (TN) equal 177

Accuracy

```
ann.evaluate(X_test2,y_test2)
```

```
80/80 [==============================] - 1s 7ms/step - loss: 0.1384 - acc: 0.9664
[0.13843630254268646, 0.9663898944854736]
```

The accuracy reached by applying ANN without SMOTE 0.9664 which is 96.64%

After calculating some findings from the confusion matrix which identify the accuracy such as

ACC = (TP + TN) / (P + N) = 0.9628 = 96.28%

Precision = TP / (TP + FP) = 0.9943= 99.43%

F1 score = 2TP / (2TP + FP + FN) = 0.9796 = 97.96%

Recall = TP / (TP + FN) = (2258) / (2258 +81) = 0.9653 = 96.5

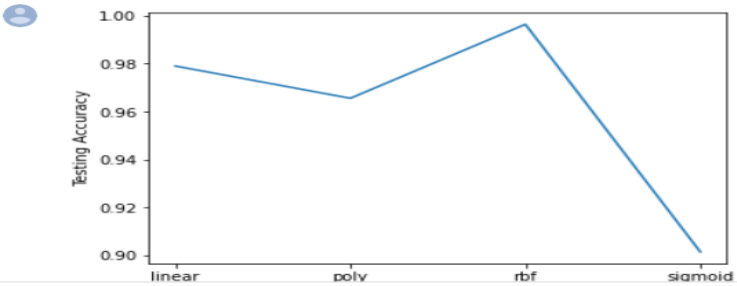
SVM (With and Without SMOTE)

SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable using supporting vector depends on kernels they use.

First of all, we test accuracy score of all kernels and plot it to find most suitable one for our data. probability is true to enable probability estimates so we can call fit. Random _state =4 which going to be used when shuffling the data for probability estimates because probability is true. Then we fit the data and got the results .

So We found that rbf is the most suitable one from the accuracy results

```
kernels = ['linear','poly','rbf','sigmoid']
scores = []
for k in kernels:
    svmm = svm.SVC(kernel=k,probability=True,random_state=4)
    svmm.fit(X_train2,y_train2)
    svmm_pred = svmm.predict(X_test2)
    scores.append(metrics.accuracy_score(y_test2, svmm_pred))
plt.plot(kernels, scores)
plt.xlabel('Kernel')
plt.ylabel('Testing Accuracy')
plt.show()
```



Then we trained the model using rbf (Max index) kernel and test prediction accuracy on of training set and test set

```
svmm =  svm.SVC(kernel=kernels[max_index],probability=True)
clf = svmm.fit(X_train2,y_train2)
svmm_pred = svmm.predict(X_test2)
print("Accuracy on training set: {:.6f}".format(svmm.score(X_train2, y_train2)))
print("Accuracy on test set: {:.6f}".format(svmm.score(X_test2, y_test2)))
```

```
Accuracy on training set: 0.996308
Accuracy on test set: 0.996441
```

We trained the model using differed prepressed data (with and without smote) and these are the results for each

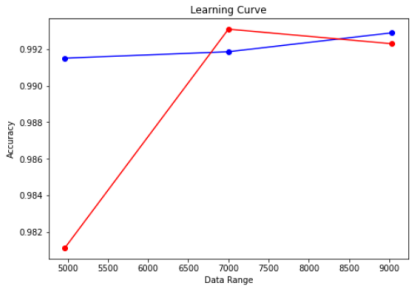
With Smote

```
[ ]
print(classification_report(y_test,svmm_pred))
```

	precision	recall	f1-score	support
0	1.00	0.99	0.99	2261
1	0.99	1.00	0.99	2256
accuracy			0.99	4517
macro avg	0.99	0.99	0.99	4517
weighted avg	0.99	0.99	0.99	4517

We made and print a classification report for model using test data

```
train_sizes_model, train_scores_model, valid_scores_model, *_ =learning_curve(svmm, smoteX, smoteY,)
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.scatter(x=train_sizes_model,y= train_scores_model.mean(axis=1), color='b')
ax.scatter(x=train_sizes_model,y=valid_scores_model.mean(axis=1), color='r')
ax.plot(train_sizes_model,train_scores_model.mean(axis=1), color='b')
ax.plot(train_sizes_model,valid_scores_model.mean(axis=1), color='r')
ax.set_xlabel('Data Range')
ax.set_ylabel('Accuracy')
ax.set_title('Learning Curve')
plt.show()
```



We draw learning curve show both of training data and test data (x_Axis = data range, Y_Axis= acc) which indicate that training curve is keep growing slowly While validation shows much lower score and keep growing with high frequency until 3000 then it starts decrease slowly so model has high variability and model could benefit from more data but it would become far less variable in the test data.

Accuracy of the model :

```
[ ]  svmm =  svm.SVC(kernel=kernels[max_index],probability=True)
     clf = svmm.fit(X_train,y_train)
     svmm_pred = svmm.predict(X_test)
     print("Accuracy on training set: {:.6f}".format(svmm.score(X_train, y_train)))
     print("Accuracy on test set: {:.6f}".format(svmm.score(X_test, y_test)))

     Accuracy on training set: 0.992620
     Accuracy on test set: 0.991809
```

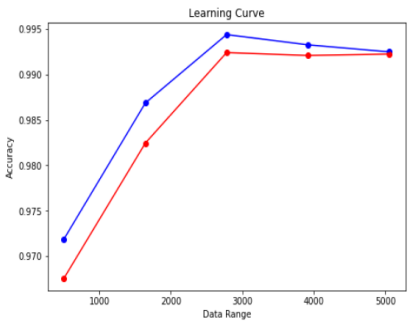
Without smote

```
[ ] print(classification_report(y_test2,svmm_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2271
1	0.98	0.99	0.98	258
accuracy			1.00	2529
macro avg	0.99	0.99	0.99	2529
weighted avg	1.00	1.00	1.00	2529

We made and print a classification report for model using test data

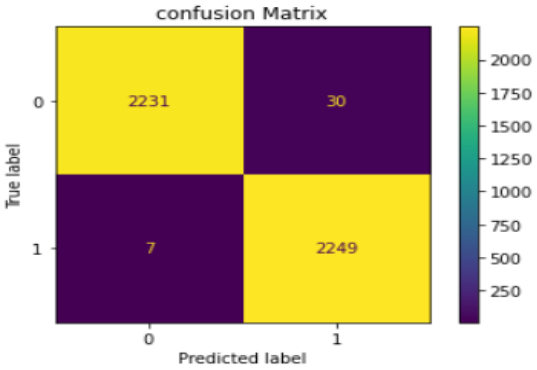
```
train_sizes_model, train_scores_model, valid_scores_model, *_ =learning_curve(svmm, X, y,)
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.scatter(x=train_sizes_model,y= train_scores_model.mean(axis=1), color='b')
ax.scatter(x=train_sizes_model,y=valid_scores_model.mean(axis=1), color='r')
ax.plot(train_sizes_model,train_scores_model.mean(axis=1), color='b')
ax.plot(train_sizes_model,valid_scores_model.mean(axis=1), color='r')
ax.set_xlabel('Data Range')
ax.set_ylabel('Accuracy')
ax.set_title('Learning Curve')
plt.show()
```



We draw learning curve show both of training data and test data (x_Axis = data range, Y_Axis= acc) which indicate that at the begging the learning curve is growing up until data range is 3000 then the data make the model over fitting (stop learning) where validation curve fit with new data until 3000 it stays as its. The difference between your train and validation scores isn't large so model learn to be good fitting

Accuracy of the model :

```
svmm =  svm.SVC(kernel=kernels[max_index],probability=True)
clf = svmm.fit(X_train2,y_train2)
svmm_pred = svmm.predict(X_test2)
print("Accuracy on training set: {:.6f}".format(svmm.score(X_train2, y_train2)))
print("Accuracy on test set: {:.6f}".format(svmm.score(X_test2, y_test2)))

Accuracy on training set: 0.996308
Accuracy on test set: 0.996441
```



Will be describe below

ACC = (TP + TN) / (P + N) = 0.9920

F1 = 2TP / (2TP + FP + FN) =

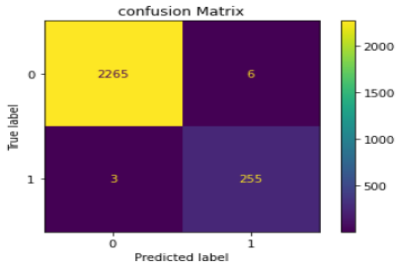
2*(Recall * Precision) / (Recall + Precision) = 0.9921

Precision = TP/ (TP + FP) = 0.9873

Recall = TP/(TP+FN) = 0.996

```
plot_confusion_matrix(svmm, X_test2, y_test2)
plt.title('confusion Matrix')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/
  warnings.warn(msg, category=FutureWarning)
```



Will be describe below

ACC = (TP + TN) / (P + N) = 0.9964

F1 = 2TP / (2TP + FP + FN) =

2*(Recall * Precision) / (Recall + Precision) = 0.9980

Precision = TP/ (TP + FP) = 0.9974

Recall = TP/(TP+FN) = 0.9986

Conclusion: comparison between the two algorithms

ANN SMOTE VS SVM SMOTE

The accuracy-based confusion matrix

Accuracy in the ANN reached 98.89% while in the SVM the accuracy was higher a little bit reaching 99.20%

And same goes for the precision (percent of correct prediction) its percentage for the ANN was 97.97% which is less than the SVM by 0.76% as it was 98.73%, moreover, the F1 score (*percent of positive predictions)* was 98.88% in the ANN while the SVM was 99.21% and the recall (*percent of the positive cases)* for both was so similar as the ANN was slightly higher as it was 99.8% in the other hand the SVM was 99.6%

ANN Without SMOTE VS SVM Without SMOTE

Accuracy of SVM model is better than Ann model where SVM accuracy = 99.64 where ANN Accuracy = 96.28.

They both got almost the same percent of correct prediction (SVM=99.7 , ANN=99.4)

SVM got more correct positive prediction than Ann where SVM= 99.80 and ANN=97.96

SVM cached more positive cases than ANN

Where SVM= 99.86 and ANN = 96.53

References:

1- “Supportvectornetworks.”Available:http://image.diku.dk/imagecanon/material/cortes_vapnik95.pdf. [Accessed: Apr-2022].
2- E. W. Saad and D. C. Wunsch, “Neural network explanation using inversion,” *Neural Networks*, vol. 20, no. 1, pp. 78–93, 2007.
3- R. Pupale, “Support vector machines(svm) - an overview,” *Medium*, 11-Feb-2019. Available: https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989.
4- https://www.sciencedirect.com/topics/engineering/confusionmatrix#:~:text=A%20confusion%20matrix%20is%20a,performance%20of%20a%20classification%20algorithm
5- https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62
6- https://scikitlearn.org/stable/autoexamples/modelselection/plotlearning_curve.html
7- http://rasbt.github.io/mlxtend/user_guide/plotting/plot_learning_curves/