Project 6 Report

Team members: Bishal Sainju, Jack Kiefer

Introduction: Fraud Detection ¶

In this project we used various predictive models to see how accurate they are in detecting whether a transaction is a normal payment or a fraud. The features provided were already scaled and the names of the features were not shown due to privacy reasons. Only 2 features are not scaled, time and amount.

Main Objective of this project is:

1. Learning to deal with imbalanced datasets?
2. Implementing various techniques like Random Under Sampling method to balance the unbalanced datasets.
3. Building models on the basis of various classifiers (Logistic Regression and SVM) and figuring out which classifier works the best.
4. Seeing if model built from Undersampling method generalize well in whole of the dataset.

For analysis we used Logistic Regression and SVC classifier.
We have found quite an interesting result that, when model is build from the subset of the original sample, since there are a lot of fraud classes in the undersample, which is generally not the case in real life, where most of the transactions are non-fraud, and because we built the model from subsample, where this property is violated, we get a model, which is biased towards frauds, i.e., it will predict most of the non-fraud transactions fraud, which is a problem, as the model that we build from this sub-sample does not quite work well with unseen data.
So, this might be a problem with the model that we created.

Introduction: Religion & Philosophy

The purpose of this analysis is to determine the predictive power of a historical society's religious configuration for whether or not the society developed a system of philosophy. This is a potentially useful metric for translators of ancient texts, as knowing the likely contexts of a given untranslated text can be very helpful for translation.

Dataset: Fraud Detection

The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
It contains only numerical input variables which are the result of a PCA transformation. However, due to confidentiality issues, the original features were not provided. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.
The total rows is 284807, of which most of them are non-fraud records, actually 99.83%. So, we need to make

our datasets balanced, so, what we did was, since, there were only 492 records, we took those, and then randomly sampled 492 non-fraud data from all the non-fraud data, and balanced our dataset. We then checked for null values, there weren't any. Then, we standardized features time and amount.

Dataset: Religion & Philosophy

This analysis utilizes the *Seshat Global History Databank* freely downloadable from seshatdatabank.info. Carefully assembled by a team of anthropologists and historians, the dataset is a large collection of quantitative, cross-cultural information on over 400 polities from 30 geographical sampling locations across thousands of years of human history and pre-history.

Significant munging was initially preformed on the dataset when it was first used for Project 4. We summarize the initial munging here:

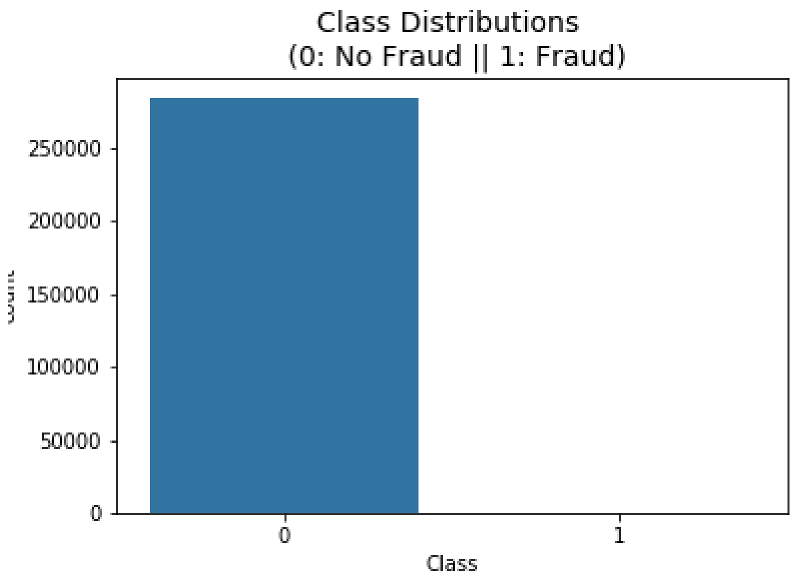
- Data points were aggregated into rows by polity
- For simplicity, only data points with a single associated date were included
- Dates were converted from BCE/CE notation to integers with 1BCE centered at 0
- Typographical and other minor human errors were corrected

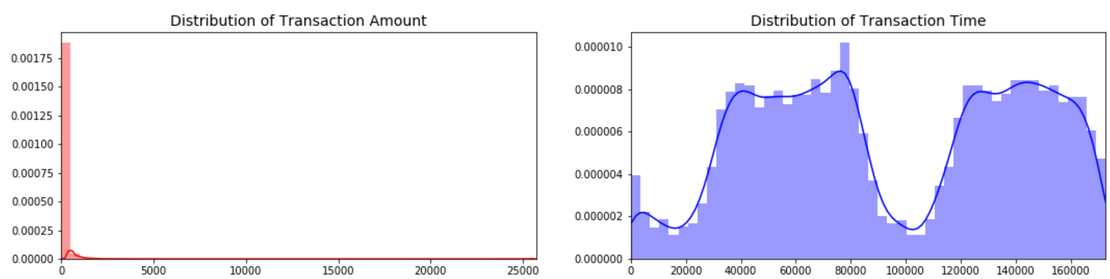
For this analysis, only minor modifications had to be made to the pre-existing munging algorithm. Namely, additional typographical errors were discovered that unnecesarily separated columns containing the same type of information. These errors were corrected.

Again, the entire munging algorithm is somewhat hefty to include in print (it is nearly 200 lines). We provide the

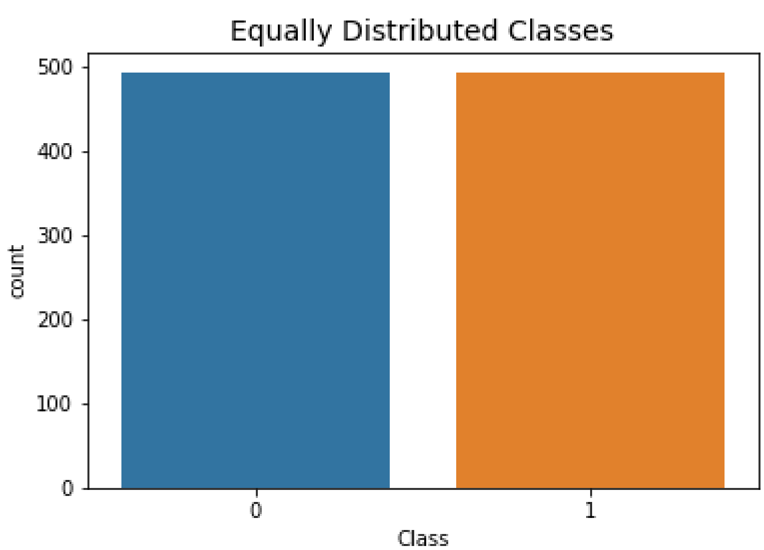
Analysis: Fraud Detection

1. Dataset is highly imbalanced.

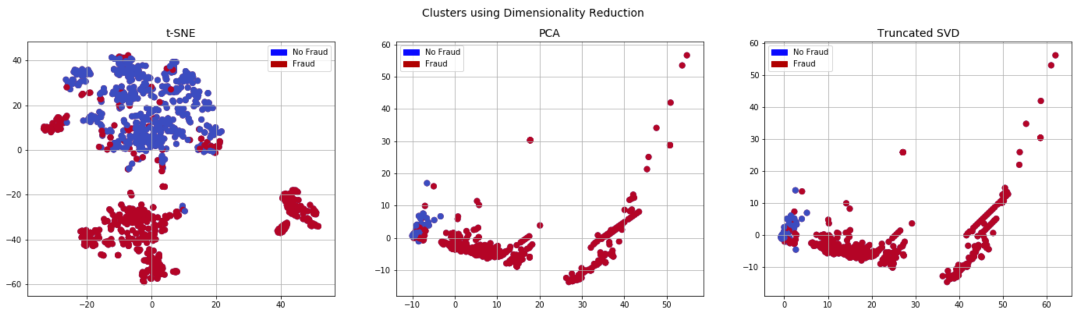




2. Since the dataset was highly imbalanced, we balanced the dataset using random under sampling method.



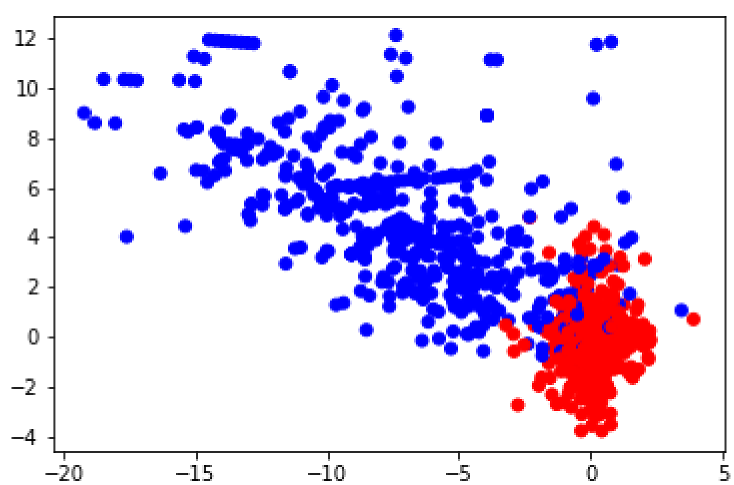
3. Then we tried to do some dimensionality reduction and cluster analysis to see if the models will perform well. Basically by this clustering we will understand if the models that we are goin to build will be able to predict well or not. So, for this purpose we basically used 3 methods: t-SNE, PCA and truncated SVD methods inorder to cluster our data. We first reduced the attributes into 2 component using each of those 3 methods and then colored different classes differently to see the separatability of the classes. And by t-SNE method we see the separatability.
   Since, t-SNE algorithm is able to detect clusters pretty accurately in every scenario, we used it, and it did pretty well in our case too.



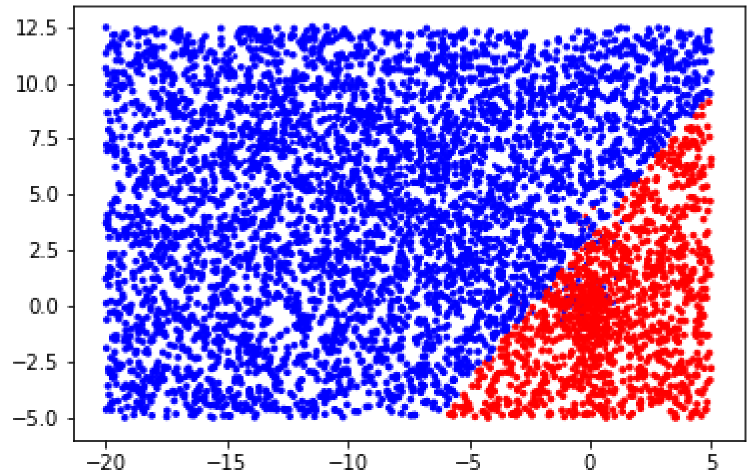
4. Then we used SelectKBest and f_classif metric to see which features were more important. Following are the features arranged in the order of their importance

```
['V14', 'V4', 'V12', 'V11', 'V10', 'V16', 'V3', 'V17', 'V9', 'V2', 'V7', 'V18', 'V1', 'V6', 'V5', 'V19', 'V20', 'V2
1', 'scaled_time', 'V28', 'V27', 'scaled_amount', 'V26', 'V8', 'V13', 'V24', 'V23', 'V25', 'V15', 'V22']
```

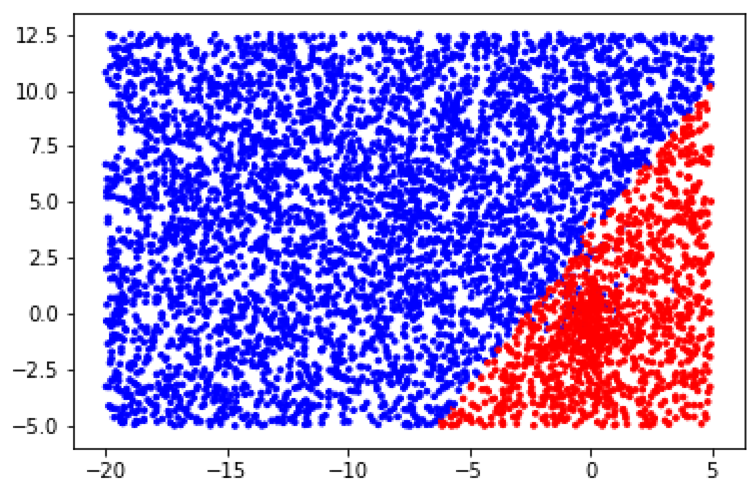
5. Then we take 2 most important features 'V14' and 'V4' and then build the model on the basis of this.
   Scatterplot of V14 and V4:



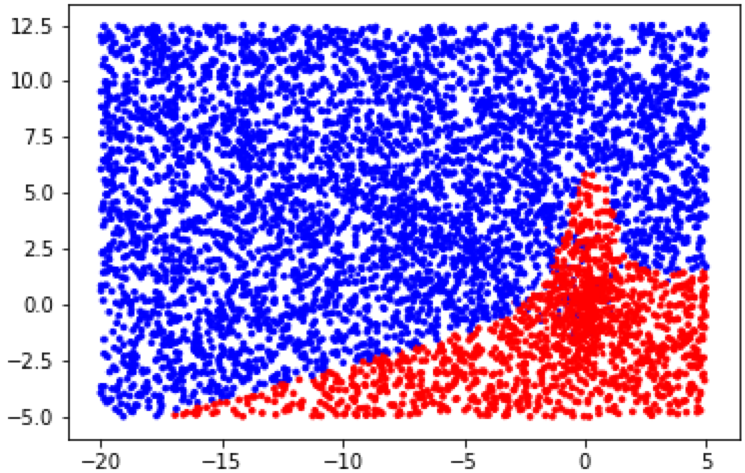
Decision boundary using Logistic Regression:



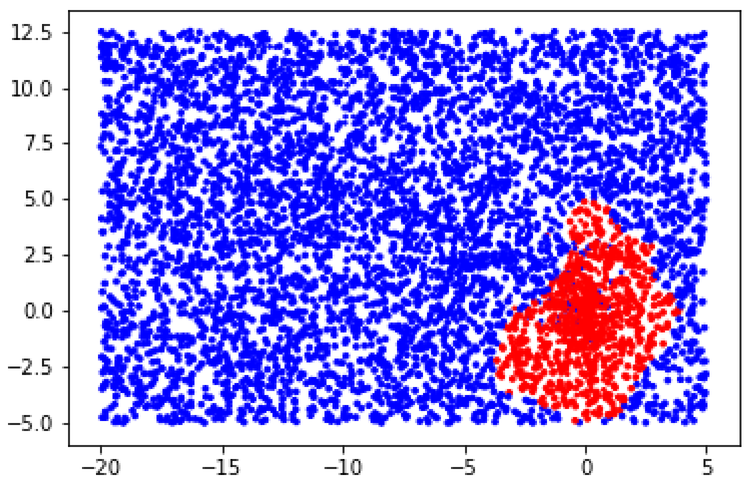
Decision boundary using Linear SVM:



Decision boundary using SVM(kernel=polynomial):



Decision boundary using SVM(kernel=RBF):



It seems like SVM with rbf kernel best separates the datasets, however, when we consider all of the parameters, our gridsearch algorithm actually will search linearSVM as the best model for the given datasets.

1. Finally we used gridSearch to search for the best parameters for both logistic regression and SVC classifier.

Analysis: Philosophy

We use a basic logistic regression to create a predictive model. This is a suitable method as we are attempting to classify the simple binary distinction of "did have system of philosophy" versus "did not have system of philosophy."

We decided on two predictive input variables that encode, respectively:

- The number of levels in the society's religious hierarchy. For example, the modern-day Catholic church is generally thought of to have five (Deacons > Priests > Bishops > Archbishops and Cardinals > The Pope)
- Whether or not the society had a writing system

Note that, while we speculate that the presence of a writing system strongly correlates with the presence of philosophy, writing is not neccesarrily a predicate for philosophy. Some societies have well-developed traditions of oral philosophy.

Results: Fraud Detection

1. The best parameter for both logistic regression and SVC classifier obtained from grid search is as follows:

```
Logistic Regression best parameter: LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)

SVC best parameter: SVC(C=0.5, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

   We see that for SVM the kernel it choses as the best one is LinearSVM.
2. Then we used the best parameters for logistic regression and used that model, and using that model we find out the following classification report.

```
Recall Score: 0.90
Precision Score: 0.97
F1 Score: 0.93
Accuracy Score: 0.93
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        99
           1       0.97      0.90      0.93        98

   micro avg       0.93      0.93      0.93       197
   macro avg       0.94      0.93      0.93       197
weighted avg       0.94      0.93      0.93       197

Confusion Matrix:
```

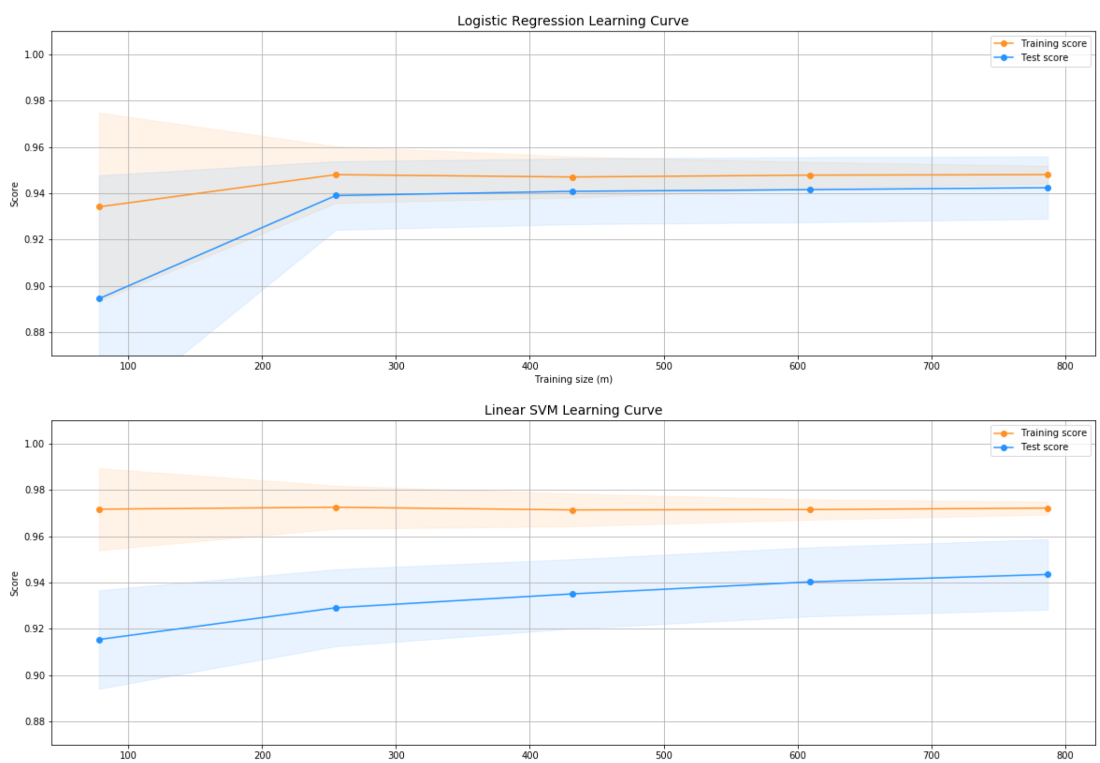
```
Out[26]: Text(0.5, 1.0, 'Logistic Regression Classifier \n Confusion Matrix')
```

3. Then we used the best parameters for logistic regression and used that model, and using that model we find out the following classification report.

```
Recall Score: 0.88
Precision Score: 0.97
F1 Score: 0.92
Accuracy Score: 0.92
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        99
           1       0.97      0.88      0.92        98

   micro avg       0.92      0.92      0.92       197
   macro avg       0.93      0.92      0.92       197
weighted avg       0.93      0.92      0.92       197

Confusion Matrix:
```

Out[27]:
```
Text(0.5, 1.0, 'Suppor Vector Classifier \n Confusion Matrix')
```

4. Then we look at the learning curve for our best logistic regression model, and best linear SVM model.



We see that logistic regression model seems to generalize better than linear svm, as there seems to be much larger gap between train error and test in case of linear svm than in logistic regression.

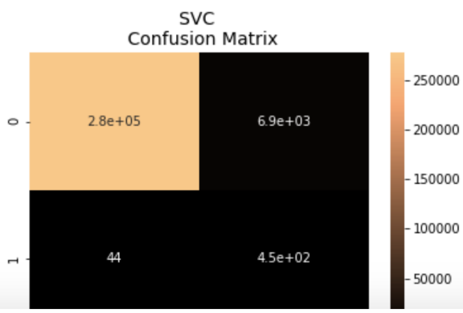
5. However, neither of the model generalizes well in the overall dataset. So, we then used these model for predicting the overall datasets classes, and we figured out that neither of them did that well. Since we showed our model even distribution of fraud and non-fraud cases, it assumed that the real world dataset

would also have even cases, and it started predicting in general cases, most of those non-fraud recors also as fraud cases.

Following is the classification report and confusion matrix of one of the model, Best Linear SVM model:

```
Recall Score: 0.91
Precision Score: 0.06
F1 Score: 0.11
Accuracy Score: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99    284315
           1       0.06      0.91      0.11       492

   micro avg       0.98      0.98      0.98    284807
   macro avg       0.53      0.94      0.55    284807
weighted avg       1.00      0.98      0.99    284807

Confusion Matrix:
```

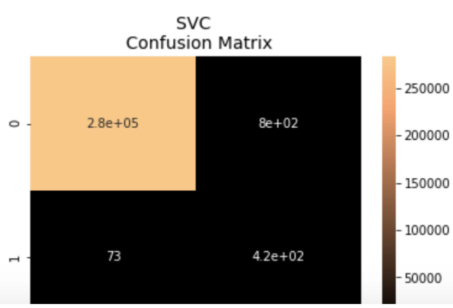
Out[65]: Text(0.5, 1.0, 'SVC \n Confusion Matrix')



6. But can we improve this score, we tried weighting class 0 ('Non-fraud') with various values from [1, 10], and for weight of 9 on class non-fraud, this actually improves the accuracy for non-fraud cases.

```
Recall Score: 0.85
Precision Score: 0.34
F1 Score: 0.49
Accuracy Score: 1.00
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    284315
           1       0.34      0.85      0.49       492

   micro avg       1.00      1.00      1.00    284807
   macro avg       0.67      0.92      0.74    284807
weighted avg       1.00      1.00      1.00    284807

Confusion Matrix:
```

Out[79]: Text(0.5, 1.0, 'SVC \n Confusion Matrix')



Results: Religion & Philosophy

In this analysis, we created a predictive model to determine whether or not a society developed a system of philosophy based on the number of levels in their religious hierarchy as well as wether or not they had a writing system. The model has surprisngly decent performance with a mean F1 Score of roughly 90%. This indicates that the two input variables (number of religious levels and the presence of a writing system) have relatively strong predictive power

Code: Fraud detection

In [131]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('creditcard.csv')
df
```

Out[131]:

	Time	V1	V2	V3	V4	V5	V6	V7
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583
10	10.0	1.449044	-1.176339	0.913860	-1.375667	-1.971383	-0.629152	-1.423236
11	10.0	0.384978	0.616109	-0.874300	-0.094019	2.924584	3.317027	0.470455
12	10.0	1.249999	-1.221637	0.383930	-1.234899	-1.485419	-0.753230	-0.689405
13	11.0	1.069374	0.287722	0.828613	2.712520	-0.178398	0.337544	-0.096717
14	12.0	-2.791855	-0.327771	1.641750	1.767473	-0.136588	0.807596	-0.422911
15	12.0	-0.752417	0.345485	2.057323	-1.468643	-1.158394	-0.077850	-0.608581
16	12.0	1.103215	-0.040296	1.267332	1.289091	-0.735997	0.288069	-0.586057
17	13.0	-0.436905	0.918966	0.924591	-0.727219	0.915679	-0.127867	0.707642
18	14.0	-5.401258	-5.450148	1.186305	1.736239	3.049106	-1.763406	-1.559738
19	15.0	1.492936	-1.029346	0.454795	-1.438026	-1.555434	-0.720961	-1.080664
20	16.0	0.694885	-1.361819	1.029221	0.834159	-1.191209	1.309109	-0.878586
21	17.0	0.962496	0.328461	-0.171479	2.109204	1.129566	1.696038	0.107712
22	18.0	1.166616	0.502120	-0.067300	2.261569	0.428804	0.089474	0.241147
23	18.0	0.247491	0.277666	1.185471	-0.092603	-1.314394	-0.150116	-0.946365
24	22.0	-1.946525	-0.044901	-0.405570	-1.013057	2.941968	2.955053	-0.063063
25	22.0	-2.074295	-0.121482	1.322021	0.410008	0.295198	-0.959537	0.543985
26	23.0	1.173285	0.353498	0.283905	1.133563	-0.172577	-0.916054	0.369025
27	23.0	1.322707	-0.174041	0.434555	0.576038	-0.836758	-0.831083	-0.264905
28	23.0	-0.414289	0.905437	1.727453	1.473471	0.007443	-0.200331	0.740228
29	23.0	1.059387	-0.175319	1.266130	1.186110	-0.786002	0.578435	-0.767084
...	...	...	...	...	...	...	...	...
284777	172764.0	2.079137	-0.028723	-1.343392	0.358000	-0.045791	-1.345452	0.227476
284778	172764.0	-0.764523	0.588379	-0.907599	-0.418847	0.901528	-0.760802	0.758545
284779	172766.0	1.975178	-0.616244	-2.628295	-0.406246	2.327804	3.664740	-0.533297

	Time	V1	V2	V3	V4	V5	V6	V7
284780	172766.0	-1.727503	1.108356	2.219561	1.148583	-0.884199	0.793083	-0.527298
284781	172766.0	-1.139015	-0.155510	1.894478	-1.138957	1.451777	0.093598	0.191353
284782	172767.0	-0.268061	2.540315	-1.400915	4.846661	0.639105	0.186479	-0.045911
284783	172768.0	-1.796092	1.929178	-2.828417	-1.689844	2.199572	3.123732	-0.270714
284784	172768.0	-0.669662	0.923769	-1.543167	-1.560729	2.833960	3.240843	0.181576
284785	172768.0	0.032887	0.545338	-1.185844	-1.729828	2.932315	3.401529	0.337434
284786	172768.0	-2.076175	2.142238	-2.522704	-1.888063	1.982785	3.732950	-1.217430
284787	172769.0	-1.029719	-1.110670	-0.636179	-0.840816	2.424360	-2.956733	0.283610
284788	172770.0	2.007418	-0.280235	-0.208113	0.335261	-0.715798	-0.751373	-0.458972
284789	172770.0	-0.446951	1.302212	-0.168583	0.981577	0.578957	-0.605641	1.253430
284790	172771.0	-0.515513	0.971950	-1.014580	-0.677037	0.912430	-0.316187	0.396137
284791	172774.0	-0.863506	0.874701	0.420358	-0.530365	0.356561	-1.046238	0.757051
284792	172774.0	-0.724123	1.485216	-1.132218	-0.607190	0.709499	-0.482638	0.548393
284793	172775.0	1.971002	-0.699067	-1.697541	-0.617643	1.718797	3.911336	-1.259306
284794	172777.0	-1.266580	-0.400461	0.956221	-0.723919	1.531993	-1.788600	0.314741
284795	172778.0	-12.516732	10.187818	-8.476671	-2.510473	-4.586669	-1.394465	-3.632516
284796	172780.0	1.884849	-0.143540	-0.999943	1.506772	-0.035300	-0.613638	0.190241
284797	172782.0	-0.241923	0.712247	0.399806	-0.463406	0.244531	-1.343668	0.929369
284798	172782.0	0.219529	0.881246	-0.635891	0.960928	-0.152971	-1.014307	0.427126
284799	172783.0	-1.775135	-0.004235	1.189786	0.331096	1.196063	5.519980	-1.518185
284800	172784.0	2.039560	-0.175233	-1.196825	0.234580	-0.008713	-0.726571	0.017050
284801	172785.0	0.120316	0.931005	-0.546012	-0.745097	1.130314	-0.235973	0.812722
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

284807 rows × 31 columns

Except for transaction time and transaction amount we do not know what other attributes are for the security purpose. Those columns have already been standardized.

In [132]:
```
df.describe()
```

Out[132]:

	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

8 rows × 31 columns

We see that mean amount is pretty low just $88.

In [133]:
```
df.shape
```

Out[133]:
```
(284807, 31)
```

In [134]:
```
df.isnull().sum().max()
```

Out[134]:
```
0
```

no null values

In [135]:
```
df.columns
```

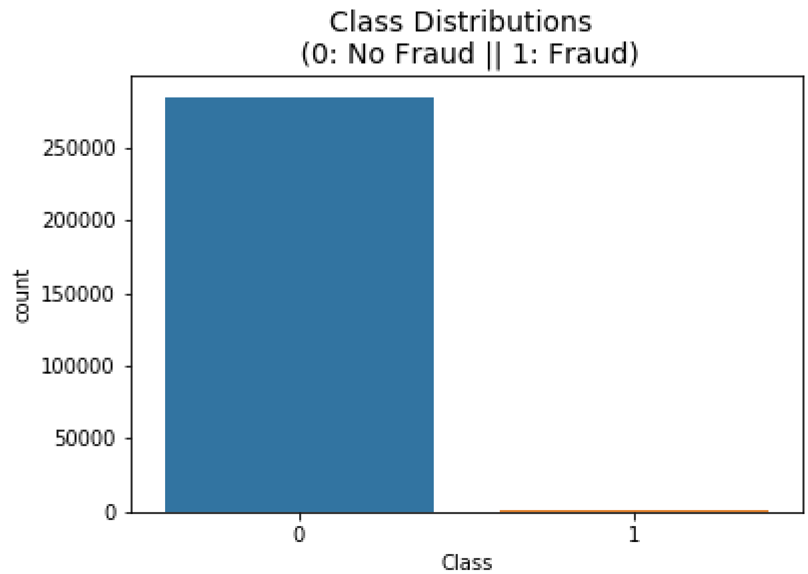
Out[135]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V
10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19',
'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amoun
t',
       'Class'],
      dtype='object')
```

In [136]:
```
print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2),
'% of the dataset')
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '%
of the dataset')
```

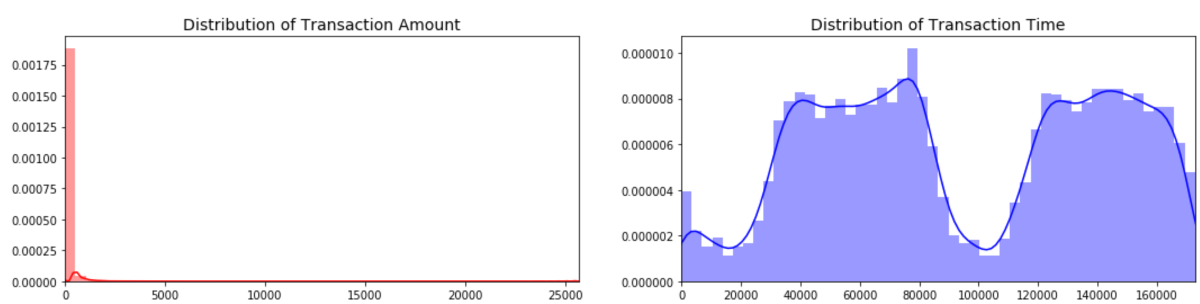
```
No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset
```

Dataset highly skewed. Wanted to learn to work with imbalanced datasets. Most of the transactions are non-fraud. If we use this dataframe as the base for our predictive models and analysis we might get a lot of errors and our algorithms will probably overfit since it will "assume" that most transactions are not fraud. But we don't want our model to assume, we want our model to detect patterns that give signs of fraud!

In [137]:
```
sns.countplot('Class', data=df)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=1
4)
plt.savefig('data/data_skew.png')
```



In [138]:
```
fig, ax = plt.subplots(1, 2, figsize=(18,4))

amount_val = df['Amount'].values
time_val = df['Time'].values

sns.distplot(amount_val, ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)
ax[0].set_xlim([min(amount_val), max(amount_val)])

sns.distplot(time_val, ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Time', fontsize=14)
ax[1].set_xlim([min(time_val), max(time_val)])
plt.savefig('data/data_distr.png')
plt.show()
```



scaling time and amount

In [139]:
```
from sklearn.preprocessing import StandardScaler, RobustScaler

# RobustScaler is less prone to outliers.

std_scaler = StandardScaler()
rob_scaler = RobustScaler()

df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.resha
pe(-1,1))
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-
1,1))

df.drop(['Time','Amount'], axis=1, inplace=True)
```

In [140]:
```
scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']

df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
df.insert(0, 'scaled_amount', scaled_amount)
df.insert(1, 'scaled_time', scaled_time)

# Amount and Time are Scaled!

df.head()
```

Out[140]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5	V6
0	1.783274	-0.994983	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388
1	-0.269825	-0.994983	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361
2	4.983721	-0.994972	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499
3	1.418291	-0.994972	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203
4	0.670579	-0.994960	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921

5 rows × 31 columns

Random Under Sampling
Basically removing the majority data to make the dataset more balanced.

In [141]:
```
df.groupby(['Class'])['Class'].count()
```

Out[141]:
```
Class
0    284315
1       492
Name: Class, dtype: int64
```

In [142]:
```
df = df.sample(frac=1)

# amount of fraud classes 492 rows.
fraud_df = df.loc[df['Class'] == 1]
non_fraud_df = df.loc[df['Class'] == 0][:492]

normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

new_df = normal_distributed_df.sample(frac=1, random_state=42)

new_df.head()
```

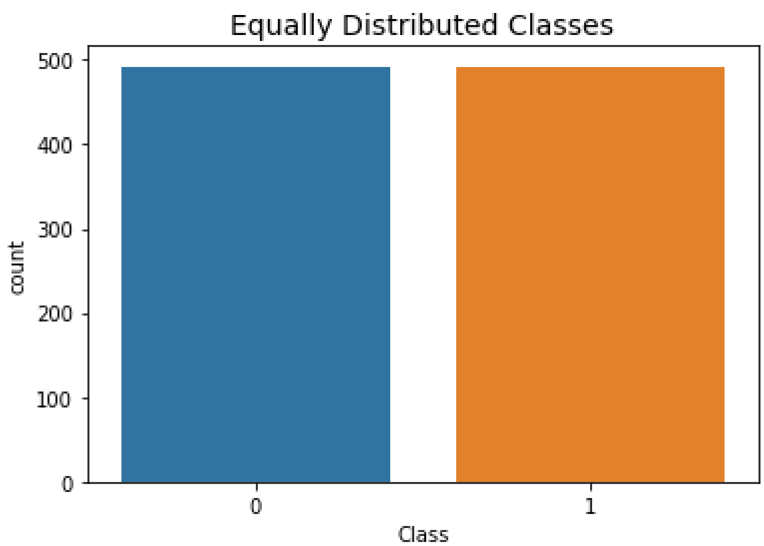
Out[142]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5	
67048	-0.083840	-0.380056	-1.265214	0.724071	2.589137	-0.378290	-0.510948	-0.243
27749	-0.041640	-0.587472	-0.860827	3.131790	-5.052968	5.420941	-2.494141	-1.811
146710	-0.237546	0.036984	-2.343709	2.101572	-1.109939	-3.065813	0.335260	-0.488
214662	1.376930	0.647035	0.467992	1.100118	-5.607145	2.204714	-0.578539	-0.174
244333	-0.293440	0.794358	-5.222968	4.641827	-8.858204	7.723502	-1.507035	-2.159

5 rows × 31 columns

In [143]:
```
print('Distribution of the Classes in the subsample dataset')
print(new_df['Class'].value_counts()/len(new_df))



sns.countplot('Class', data=new_df)
plt.title('Equally Distributed Classes', fontsize=14)
plt.savefig('data/data_noskew.png')
plt.show()
```

```
Distribution of the Classes in the subsample dataset
1    0.5
0    0.5
Name: Class, dtype: float64
```

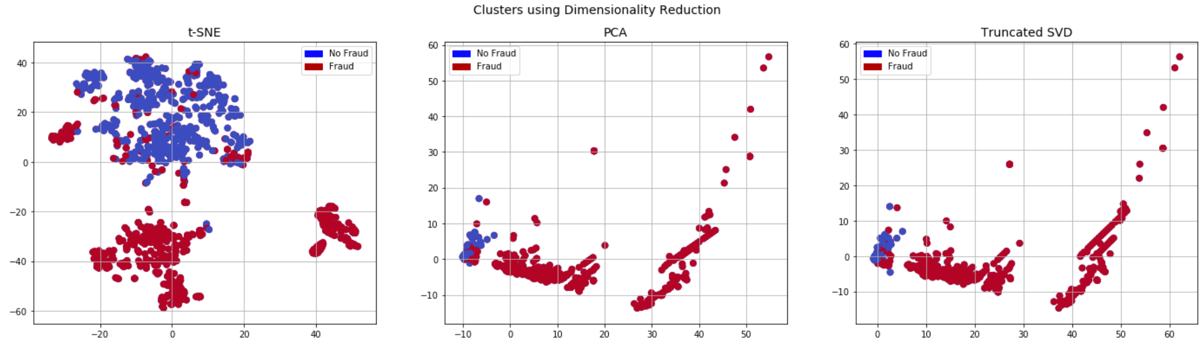


Let's see if we can cluster the datasets, thus, indicating if the predictive models will perform pretty well in separating fraud cases from non-fraud cases.

In [144]:
```
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA, TruncatedSVD
import time

X = new_df.drop('Class', axis=1)
y = new_df['Class']


# T-SNE Implementation
t0 = time.time()
X_reduced_tsne = TSNE(n_components=2, random_state=42).fit_transform(X.v
alues)
t1 = time.time()
print("T-SNE took {:.2} s".format(t1 - t0))

# PCA Implementation
t0 = time.time()
X_reduced_pca = PCA(n_components=2, random_state=42).fit_transform(X.val
ues)
t1 = time.time()
print("PCA took {:.2} s".format(t1 - t0))

# TruncatedSVD
t0 = time.time()
X_reduced_svd = TruncatedSVD(n_components=2, algorithm='randomized', ran
dom_state=42).fit_transform(X.values)
t1 = time.time()
print("Truncated SVD took {:.2} s".format(t1 - t0))
```

```
T-SNE took 5.8 s
PCA took 0.0031 s
Truncated SVD took 0.0018 s
```

In [145]:
```
import matplotlib.patches as mpatches

f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(24,6))
# labels = ['No Fraud', 'Fraud']
f.suptitle('Clusters using Dimensionality Reduction', fontsize=14)


blue_patch = mpatches.Patch(color='#0A0AFF', label='No Fraud')
red_patch = mpatches.Patch(color='#AF0000', label='Fraud')


# t-SNE scatter plot
ax1.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 0), cmap=
'coolwarm', label='No Fraud', linewidths=2)
ax1.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 1), cmap=
'coolwarm', label='Fraud', linewidths=2)
ax1.set_title('t-SNE', fontsize=14)

ax1.grid(True)

ax1.legend(handles=[blue_patch, red_patch])


# PCA scatter plot
ax2.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y == 0), cmap='co
olwarm', label='No Fraud', linewidths=2)
ax2.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y == 1), cmap='co
olwarm', label='Fraud', linewidths=2)
ax2.set_title('PCA', fontsize=14)

ax2.grid(True)

ax2.legend(handles=[blue_patch, red_patch])

# TruncatedSVD scatter plot
ax3.scatter(X_reduced_svd[:,0], X_reduced_svd[:,1], c=(y == 0), cmap='co
olwarm', label='No Fraud', linewidths=2)
ax3.scatter(X_reduced_svd[:,0], X_reduced_svd[:,1], c=(y == 1), cmap='co
olwarm', label='Fraud', linewidths=2)
ax3.set_title('Truncated SVD', fontsize=14)

ax3.grid(True)

ax3.legend(handles=[blue_patch, red_patch])
plt.savefig('data/cluster.png')
plt.show()
```



Let's see feature importance.

In [146]:
```
cols = X.columns
print(cols)
```

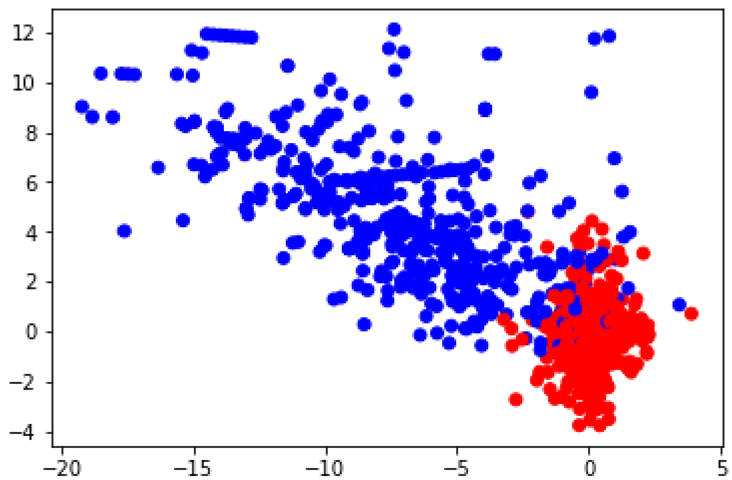
```
Index(['scaled_amount', 'scaled_time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V
6',
       'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V1
6',
       'V17', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25',
'V26',
       'V27', 'V28'],
      dtype='object')
```

In [147]:
```
from sklearn.feature_selection import SelectKBest, f_classif
k_best = SelectKBest(f_classif, k=len(cols)).fit(X, y)
score = np.array(k_best.scores_)
rank = score.argsort()[-len(cols):][::-1]
features = []
for i in rank:
    features.append(cols[i])
print(features)
```

```
['V14', 'V4', 'V11', 'V12', 'V10', 'V16', 'V3', 'V17', 'V9', 'V2', 'V
7', 'V18', 'V1', 'V6', 'V5', 'V19', 'V20', 'scaled_time', 'V21', 'V28',
'V27', 'scaled_amount', 'V8', 'V26', 'V13', 'V22', 'V15', 'V23', 'V25',
'V24']
```

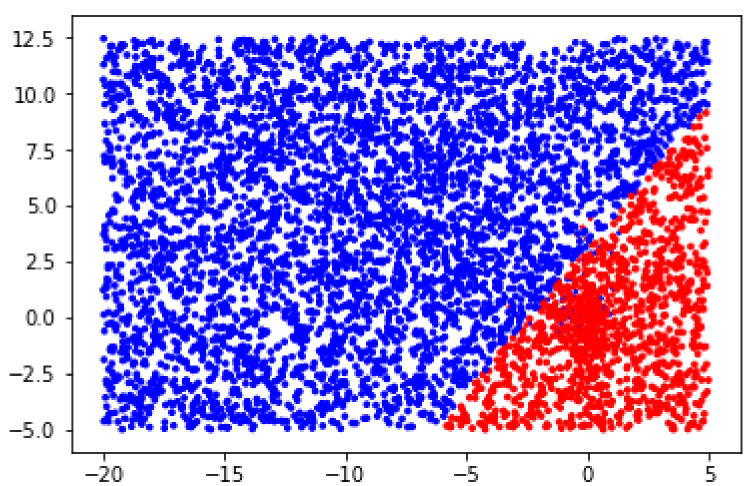
We see that V14 and V4 are the most predictive attributes in predicting the fraud. So lets take these 2 variables, and then, classify using these 2 variables only.

In [148]:
```
X = new_df[['V14', 'V4']]
y = new_df.Class

color = ['r' if y_ == 0 else 'b' for y_ in y]
plt.scatter(new_df.V14, new_df.V4, c=color)
plt.savefig('data/data_2.png')
```



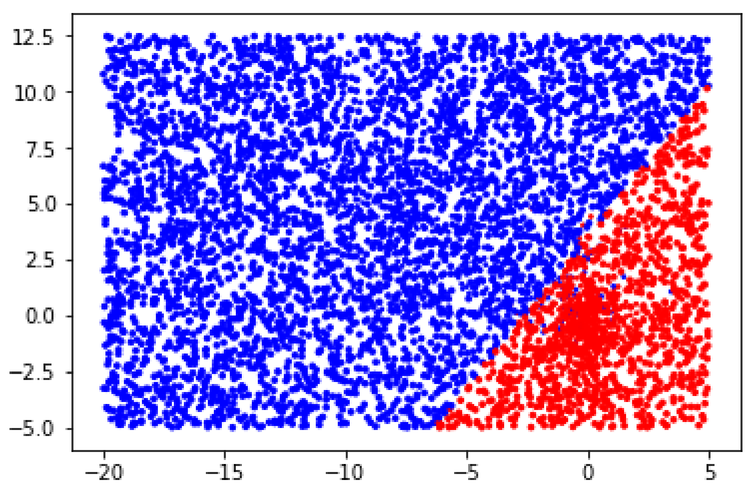
In [149]:
```
from sklearn import svm
from sklearn.linear_model import LogisticRegression
import numpy as np
from sklearn.metrics import precision_recall_fscore_support

clf = LogisticRegression()
clf.fit(X, y)

y_pred = clf.predict(X)
p,r,f,s = precision_recall_fscore_support(y, y_pred)
display('precision = {}'.format(p))
display('recall = {}'.format(r))
display('f-score = {}'.format(f))

color = ['r' if y_ == 0 else 'b' for y_ in y]
plt.scatter(new_df.V14, new_df.V4, c=color, s=3)

# add random points
import random
newx = []
newy = []
newlabel = []
for _ in range(5000):
# for _ in range(5000):
    px = random.uniform(-20,5)
    py = random.uniform(-5,12.5)
    plabel = clf.predict([[px,py]])
    newx.append(px)
    newy.append(py)
    newlabel.append(plabel)

color = ['r' if y_ == 0 else 'b' for y_ in newlabel]
plt.scatter(newx, newy, c=color, marker='o', s=7);
plt.savefig('data/log_db.png')
```

```
'precision = [0.91030534 0.9673913 ]'

'recall = [0.9695122  0.90447154]'

'f-score = [0.93897638 0.93487395]'
```



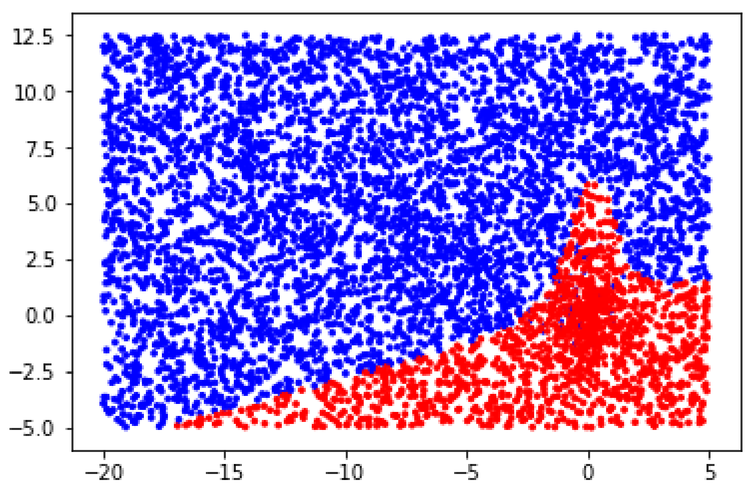
In [150]:
```
clf = svm.SVC(kernel='linear', class_weight={0:1})
clf.fit(X, y)

y_pred = clf.predict(X)
p,r,f,s = precision_recall_fscore_support(y, y_pred)
display('precision = {}'.format(p))
display('recall = {}'.format(r))
display('f-score = {}'.format(f))

color = ['r' if y_ == 0 else 'b' for y_ in y]
plt.scatter(new_df.V14, new_df.V4, c=color, s=3)

# add random points
import random
newx = []
newy = []
newlabel = []
for _ in range(5000):
# for _ in range(5000):
    px = random.uniform(-20,5)
    py = random.uniform(-5,12.5)
    plabel = clf.predict([[px,py]])
    newx.append(px)
    newy.append(py)
    newlabel.append(plabel)

color = ['r' if y_ == 0 else 'b' for y_ in newlabel]
plt.scatter(newx, newy, c=color, marker='o', s=7);
plt.savefig('data/lsvm_db.png')
```

```
'precision = [0.90619137 0.98004435]'

'recall = [0.98170732 0.89837398]'

'f-score = [0.94243902 0.93743372]'
```



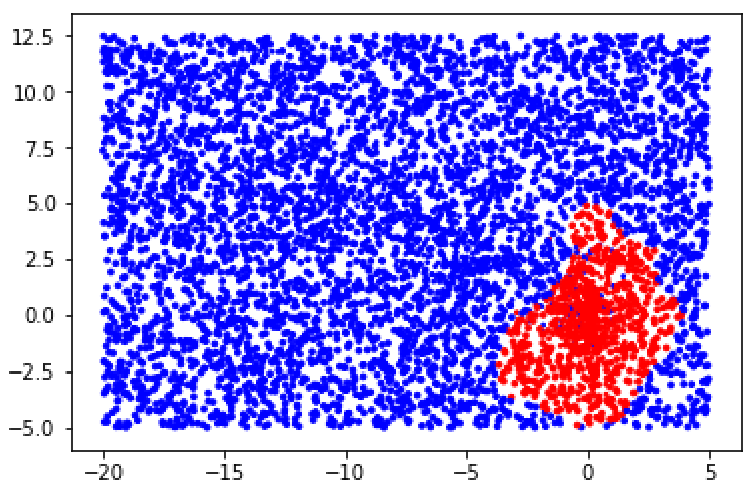
In [151]:
```
clf = svm.SVC(kernel='poly', degree=3)
clf.fit(X, y)

y_pred = clf.predict(X)
p,r,f,s = precision_recall_fscore_support(y, y_pred)
display('precision = {}'.format(p))
display('recall = {}'.format(r))
display('f-score = {}'.format(f))

color = ['r' if y_ == 0 else 'b' for y_ in y]
plt.scatter(new_df.V14, new_df.V4, c=color, s=3)

# add random points
import random
newx = []
newy = []
newlabel = []
for _ in range(5000):
# for _ in range(5000):
    px = random.uniform(-20,5)
    py = random.uniform(-5,12.5)
    plabel = clf.predict([[px,py]])
    newx.append(px)
    newy.append(py)
    newlabel.append(plabel)

color = ['r' if y_ == 0 else 'b' for y_ in newlabel]
plt.scatter(newx, newy, c=color, marker='o', s=7);
plt.savefig('data/poly_db.png')
```

```
'precision = [0.90352505 0.98876404]'

'recall = [0.9898374  0.89430894]'

'f-score = [0.94471387 0.93916756]'
```



In [152]:
```
clf = svm.SVC(kernel='rbf')
clf.fit(X, y)

y_pred = clf.predict(X)
p,r,f,s = precision_recall_fscore_support(y, y_pred)
display('precision = {}'.format(p))
display('recall = {}'.format(r))
display('f-score = {}'.format(f))

color = ['r' if y_ == 0 else 'b' for y_ in y]
plt.scatter(new_df.V14, new_df.V4, c=color, s=3)

# add random points
import random
newx = []
newy = []
newlabel = []
for _ in range(5000):
# for _ in range(5000):
    px = random.uniform(-20,5)
    py = random.uniform(-5,12.5)
    plabel = clf.predict([[px,py]])
    newx.append(px)
    newy.append(py)
    newlabel.append(plabel)

color = ['r' if y_ == 0 else 'b' for y_ in newlabel]
plt.scatter(newx, newy, c=color, marker='o', s=7);
plt.savefig('data/rbf_db.png')
```

```
'precision = [0.90841121 0.98663697]'

'recall = [0.98780488 0.9004065 ]'

'f-score = [0.94644596 0.94155154]'
```



In [153]:
```
# Use GridSearchCV to find the best parameters.
from sklearn.model_selection import GridSearchCV

X = new_df.drop('Class', axis=1)
y = new_df['Class']


# Logistic Regression
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10
, 100, 1000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg.fit(X, y)
# We automatically get the logistic regression with the best parameters.
log_reg = grid_log_reg.best_estimator_

# Support Vector Classifier
svc_params = {'C': [0.5, 0.7, 0.9, 1], 'kernel': ['rbf', 'poly', 'sigmoi
d', 'linear']}
grid_svc = GridSearchCV(svm.SVC(), svc_params)
grid_svc.fit(X, y)
# SVC best estimator
svc = grid_svc.best_estimator_


print('Logistic Regression best parameter: ' + str(log_reg))
print()
print('SVC best parameter: ' + str(svc))
```

```
Logistic Regression best parameter: LogisticRegression(C=0.1, class_wei
ght=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)

SVC best parameter: SVC(C=0.9, cache_size=200, class_weight=None, coef0
=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='poly', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [154]:
```
from sklearn.metrics import precision_score, recall_score, f1_score, roc
_auc_score, accuracy_score, \
                classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score

log_reg_score = cross_val_score(log_reg, X, y, cv=5)
print('Logistic Regression Cross Validation Score: ', round(log_reg_scor
e.mean() * 100, 2).astype(str) + '%')
print('Logistic Regression Classification Report: ')

svc_score = cross_val_score(svc, X, y, cv=5)
print('Support Vector Classifier Cross Validation Score', round(svc_scor
e.mean() * 100, 2).astype(str) + '%')
print('Support Vector Classifier Classification Report: ')
```

```
Logistic Regression Cross Validation Score:  94.61%
Logistic Regression Classification Report:
Support Vector Classifier Cross Validation Score 94.51%
Support Vector Classifier Classification Report:
```

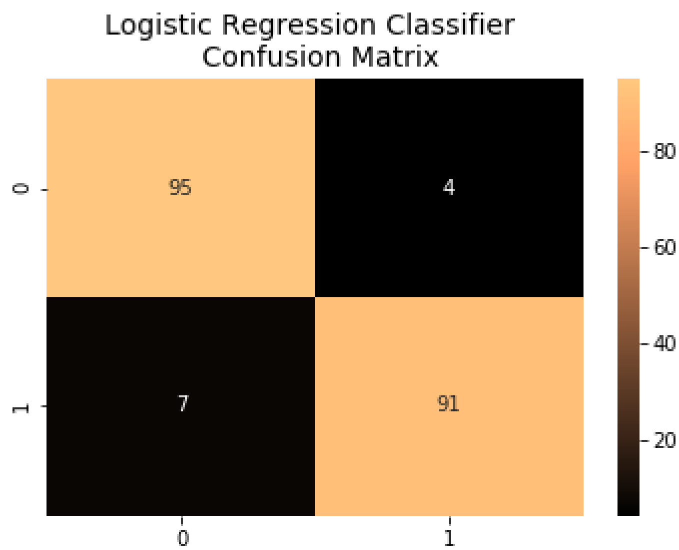
In [155]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
stratify=y, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(787, 30)
(197, 30)
(787,)
(197,)
```

In [156]:
```
y_pred = log_reg.fit(X_train, y_train).predict(X_test)
print('Recall Score: {:.2f}'.format(recall_score(y_test, y_pred)))
print('Precision Score: {:.2f}'.format(precision_score(y_test, y_pred)))
print('F1 Score: {:.2f}'.format(f1_score(y_test, y_pred)))
print('Accuracy Score: {:.2f}'.format(accuracy_score(y_test, y_pred)))
print('Classification Report: ')
print(classification_report(y_test, y_pred))
print('Confusion Matrix: ')
log_reg_cf = confusion_matrix(y_test, y_pred)
sns.heatmap(log_reg_cf, annot=True, cmap=plt.cm.copper)
plt.title("Logistic Regression Classifier \n Confusion Matrix", fontsize
=14)
```

```
Recall Score: 0.93
Precision Score: 0.96
F1 Score: 0.94
Accuracy Score: 0.94
Classification Report: 
              precision    recall  f1-score   support

           0       0.93      0.96      0.95        99
           1       0.96      0.93      0.94        98

   micro avg       0.94      0.94      0.94       197
   macro avg       0.94      0.94      0.94       197
weighted avg       0.94      0.94      0.94       197

Confusion Matrix: 
```

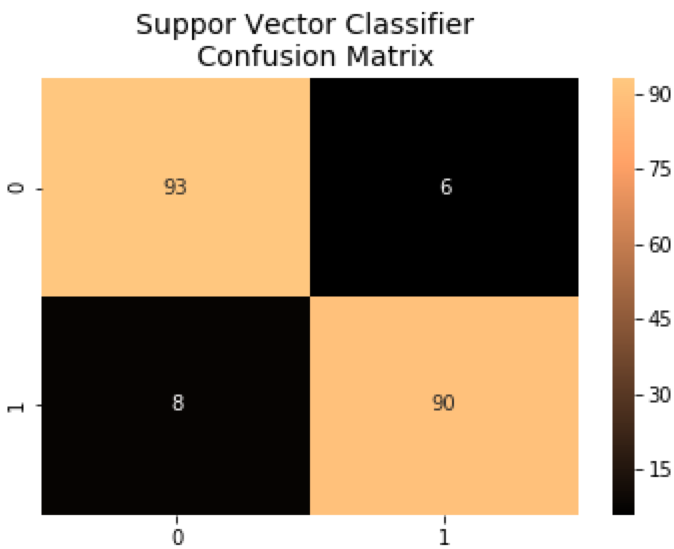
Out[156]:
```
Text(0.5, 1.0, 'Logistic Regression Classifier \n Confusion Matrix')
```



In [157]:
```
y_pred = svc.fit(X_train, y_train).predict(X_test)
print('Recall Score: {:.2f}'.format(recall_score(y_test, y_pred)))
print('Precision Score: {:.2f}'.format(precision_score(y_test, y_pred)))
print('F1 Score: {:.2f}'.format(f1_score(y_test, y_pred)))
print('Accuracy Score: {:.2f}'.format(accuracy_score(y_test, y_pred)))
print('Classification Report: ')
print(classification_report(y_test, y_pred))
print('Confusion Matrix: ')
svc_cf = confusion_matrix(y_test, y_pred)
sns.heatmap(svc_cf, annot=True, cmap=plt.cm.copper)
plt.title("Suppor Vector Classifier \n Confusion Matrix", fontsize=14)
```

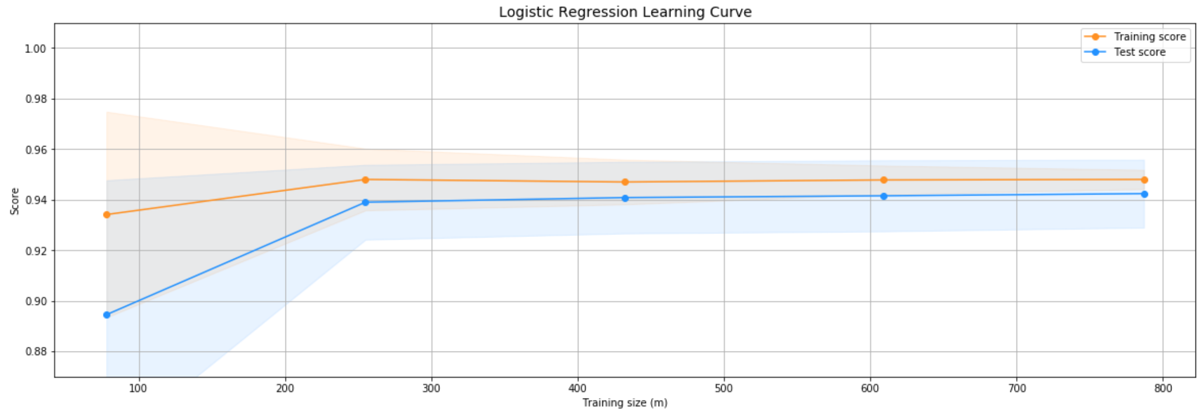
```
Recall Score: 0.92
Precision Score: 0.94
F1 Score: 0.93
Accuracy Score: 0.93
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93        99
           1       0.94      0.92      0.93        98

   micro avg       0.93      0.93      0.93       197
   macro avg       0.93      0.93      0.93       197
weighted avg       0.93      0.93      0.93       197

Confusion Matrix:
```

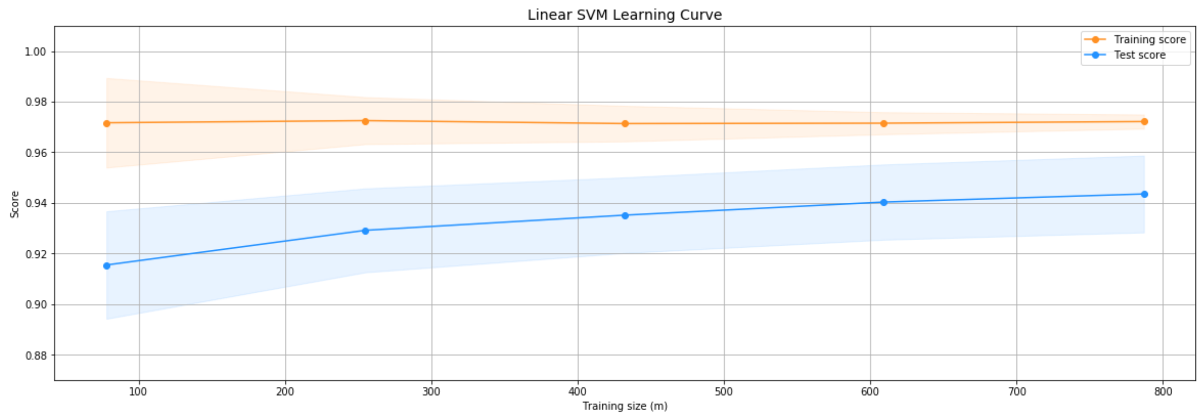
Out[157]:
```
Text(0.5, 1.0, 'Suppor Vector Classifier \n Confusion Matrix')
```



In [158]:
```
# Let's Plot LogisticRegression Learning Curve
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator1, estimator2, X, y, ylim=None, cv=None
,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    f, ((ax1, ax2)) = plt.subplots(2,1, figsize=(20,14), sharey=True)
    if ylim is not None:
        plt.ylim(*ylim)
    # First Estimator
    train_sizes, train_scores, test_scores = learning_curve(
        estimator1, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    ax1.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="#ff9124")
    ax1.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, colo
r="#2492ff")
    ax1.plot(train_sizes, train_scores_mean, 'o-', color="#ff9124",
             label="Training score")
    ax1.plot(train_sizes, test_scores_mean, 'o-', color="#2492ff",
             label="Test score")
    ax1.set_title("Logistic Regression Learning Curve", fontsize=14)
    ax1.set_xlabel('Training size (m)')
    ax1.set_ylabel('Score')
    ax1.grid(True)
    ax1.legend(loc="best")

    # Second Estimator
    train_sizes, train_scores, test_scores = learning_curve(
        estimator2, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    ax2.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="#ff9124")
    ax2.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, colo
r="#2492ff")
    ax2.plot(train_sizes, train_scores_mean, 'o-', color="#ff9124",
             label="Training score")
    ax2.plot(train_sizes, test_scores_mean, 'o-', color="#2492ff",
             label="Test score")
    ax2.set_title("Linear SVM Learning Curve", fontsize=14)
    ax2.set_xlabel('Training size (m)')
    ax2.set_ylabel('Score')
    ax2.grid(True)
    ax2.legend(loc="best")
    return plt
```

In [159]:
```
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=5)
plot_learning_curve(log_reg, svc, X, y, (0.87, 1.01), cv=cv, n_jobs=4)
plt.savefig('data/learning_curve.png')
```





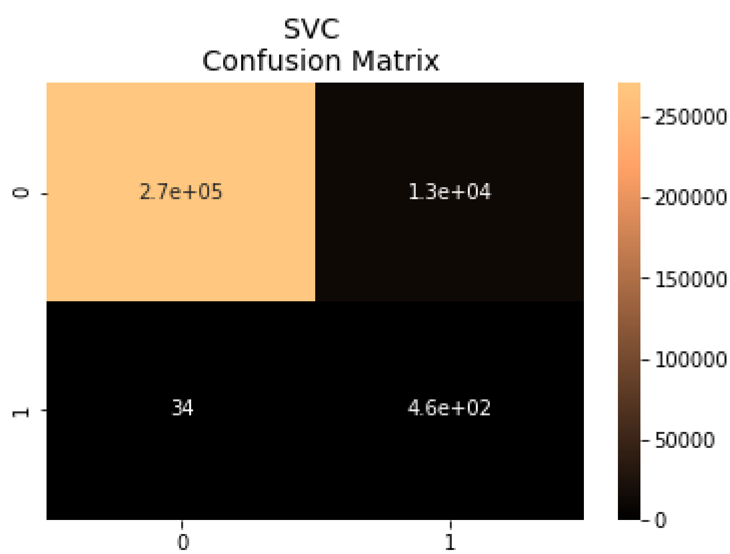
In [160]:
```
model = svm.SVC(kernel='linear', C=0.5).fit(X_train, y_train)
```

In [161]:
```
X = df.drop('Class', axis=1)
y = df['Class']

y_pred = model.predict(X)
```

In [162]:
```
print('Recall Score: {:.2f}'.format(recall_score(y, y_pred)))
print('Precision Score: {:.2f}'.format(precision_score(y, y_pred)))
print('F1 Score: {:.2f}'.format(f1_score(y, y_pred)))
print('Accuracy Score: {:.2f}'.format(accuracy_score(y, y_pred)))
print('Classification Report: ')
print(classification_report(y, y_pred))
print('Confusion Matrix: ')
svc_cf = confusion_matrix(y, y_pred)
sns.heatmap(svc_cf, annot=True, cmap=plt.cm.copper)
plt.title("SVC \n Confusion Matrix", fontsize=14)
```

```
Recall Score: 0.93
Precision Score: 0.03
F1 Score: 0.06
Accuracy Score: 0.95
Classification Report: 
              precision    recall  f1-score   support

           0       1.00      0.95      0.98    284315
           1       0.03      0.93      0.06       492

   micro avg       0.95      0.95      0.95    284807
   macro avg       0.52      0.94      0.52    284807
weighted avg       1.00      0.95      0.97    284807

Confusion Matrix: 
```

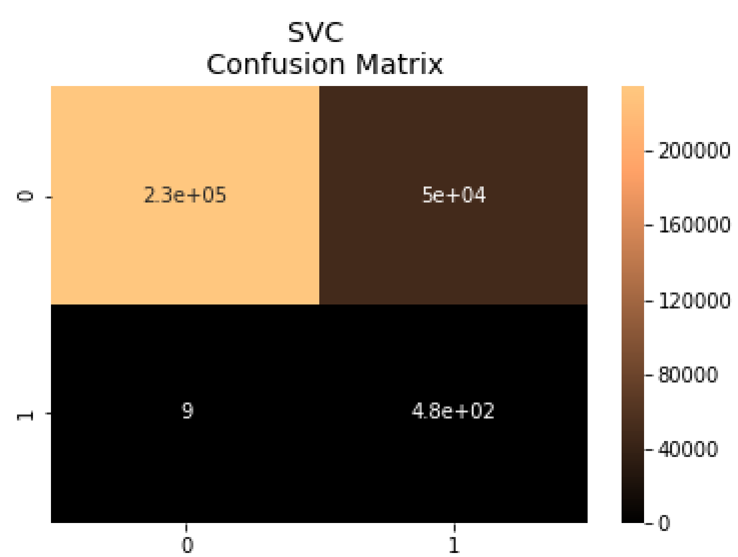
Out[162]:
```
Text(0.5, 1.0, 'SVC \n Confusion Matrix')
```



In [169]:
```
model = svm.SVC(kernel='linear', C=0.5, class_weight={1: 10}).fit(X_trai
n, y_train)

X = df.drop('Class', axis=1)
y = df['Class']

y_pred = model.predict(X)

print('Recall Score: {:.2f}'.format(recall_score(y, y_pred)))
print('Precision Score: {:.2f}'.format(precision_score(y, y_pred)))
print('F1 Score: {:.2f}'.format(f1_score(y, y_pred)))
print('Accuracy Score: {:.2f}'.format(accuracy_score(y, y_pred)))
print('Classification Report: ')
print(classification_report(y, y_pred))
print('Confusion Matrix: ')
svc_cf = confusion_matrix(y, y_pred)
sns.heatmap(svc_cf, annot=True, cmap=plt.cm.copper)
plt.title("SVC \n Confusion Matrix", fontsize=14)
```

```
Recall Score: 0.98
Precision Score: 0.01
F1 Score: 0.02
Accuracy Score: 0.83
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.82      0.90    284315
           1       0.01      0.98      0.02       492

   micro avg       0.83      0.83      0.83    284807
   macro avg       0.50      0.90      0.46    284807
weighted avg       1.00      0.83      0.90    284807

Confusion Matrix:
```

Out[169]:
```
Text(0.5, 1.0, 'SVC \n Confusion Matrix')
```



Code: Religion & Philosophy

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_fscore_support
from sklearn.linear_model import LogisticRegression

# Given a dataframe and a list of columns,
# return a dataframe containing only the
# Series which have complete data for those columns.
# The returned dataframe only contains the columns
# specified.
def squash(df, columns):
    columns.append('Polity')
    squashed = df.copy()
    squashed = squashed[columns]
    squashed = squashed.dropna()
    squashed = squashed.set_index('Polity')
    return squashed

# Read in data
seshat = pd.read_csv('seshat.csv')

# Define explanatory variables
xvar = 'Religious levels'
yvar = 'Philosophy'

# Get entries with only these variables
seshat = squash(seshat, [xvar, yvar, 'Script'])

# Define x and y axes
x = seshat[['Religious levels', 'Script']]
y = seshat.Philosophy

# Fit!
lm = LogisticRegression(solver='lbfgs')
lm.fit(x,y)

# Predict
y_pred = lm.predict(x)

print('predicted range: [{0:.2f},{1:.2f}]'.format(min(y_pred), max(y_pre
d)))

# Evaluate
p,r,f,s = precision_recall_fscore_support(y, y_pred)
print('precision = {}'.format(p))
print('recall = {}'.format(r))
print('fscore = {}'.format(f))

# Plot
ax = plt.gca()
plt.plot(y_pred)
plt.xlabel(xvar)
plt.ylabel(yvar)
ax.set_xlim([0,11])
plt.show()
```