In [40]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
claim=pd.read_csv("C:/claims_management/train.csv")
```

In [3]:

```
print('Proceed', round(claim['target'].value_counts()[1]/len(claim) * 100,2), '% of the
dataset')
print('Paper Works Remain of', round(claim['target'].value_counts()[0]/len(claim) * 100
,2), '% of the dataset')
```

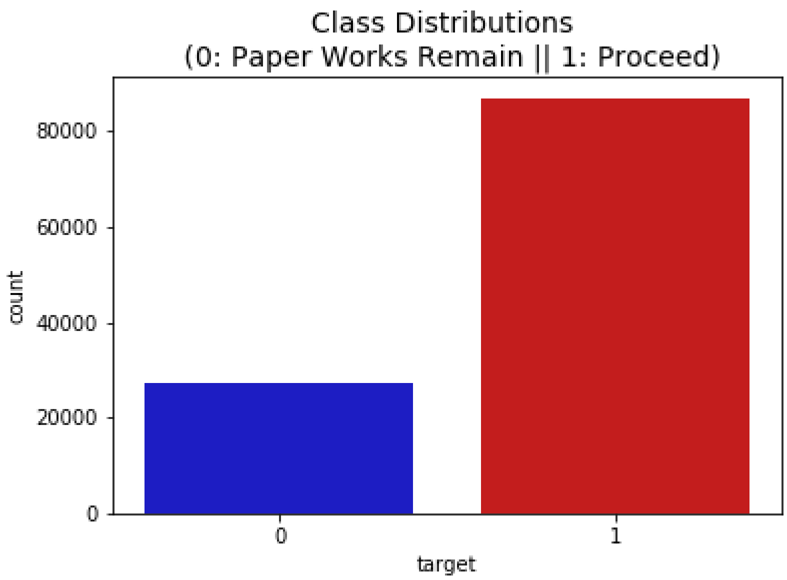
```
Proceed 76.12 % of the dataset
Paper Works Remain of 23.88 % of the dataset
```

In [4]:

```
colors = ["#0101DF", "#DF0101"]

sns.countplot('target', data=claim, palette=colors)
plt.title('Class Distributions \n (0: Paper Works Remain || 1: Proceed)', fontsize=14)
```

Out[4]:

```
Text(0.5, 1.0, 'Class Distributions \n (0: Paper Works Remain || 1: Procee
d)')
```



In [5]:

```
def info_null(df):
    total_null = df.isnull().sum().sort_values(ascending = False)
    percentage = round(total_null/len(claim) * 100, 2)
    result = pd.concat([total_null, percentage], axis = 1, keys = ['total Null Values',
'Percentage'])
    result.sort_values(by = ['total Null Values', 'Percentage'], ascending = False)
    return(result[80:100])
```

In [6]:

```
info_null(claim)
```

Out[6]:

| | total Null Values | Percentage |
|---|---|---|
| v4 | 49796 | 43.56 |
| v76 | 49796 | 43.56 |
| v2 | 49796 | 43.56 |
| v87 | 48663 | 42.57 |
| v105 | 48658 | 42.56 |
| v98 | 48654 | 42.56 |
| v70 | 48636 | 42.54 |
| v128 | 48624 | 42.53 |
| v5 | 48624 | 42.53 |
| v36 | 48624 | 42.53 |
| v82 | 48624 | 42.53 |
| v81 | 48624 | 42.53 |
| v117 | 48624 | 42.53 |
| v109 | 48624 | 42.53 |
| v108 | 48624 | 42.53 |
| v89 | 48619 | 42.53 |
| v124 | 48619 | 42.53 |
| v25 | 48619 | 42.53 |
| v54 | 48619 | 42.53 |
| v63 | 48619 | 42.53 |

In [7]:

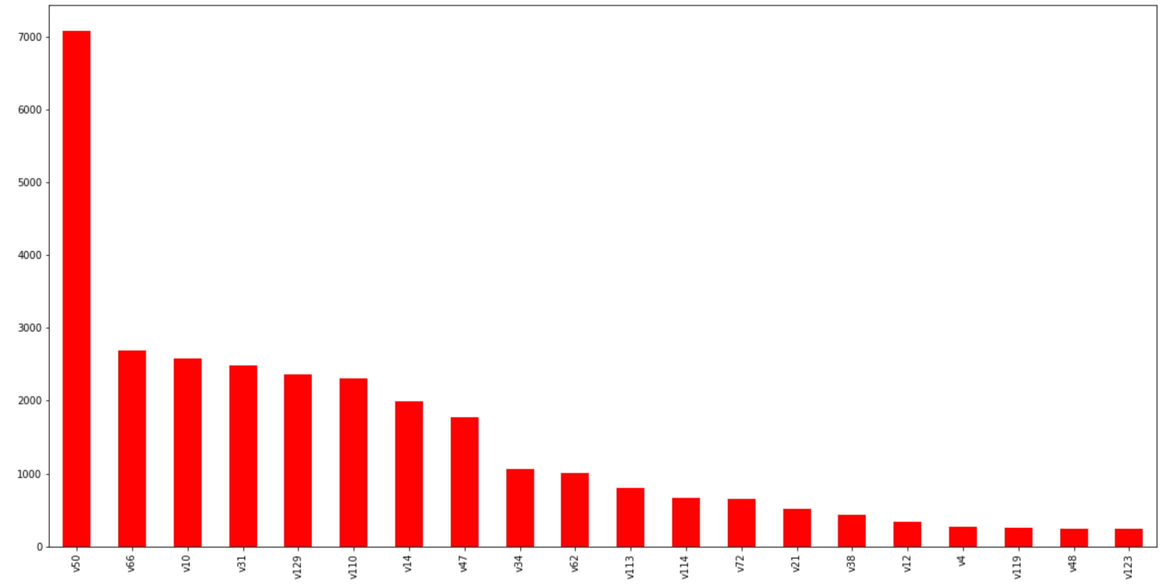
```
disc=claim.select_dtypes(include = 'object')
```

In [8]:

```
for _ in disc.columns:
    k = {i:d for d, i in enumerate(claim[_].unique())}
    claim[_].replace(k, inplace = True)
for i in claim.columns:
    claim[i].fillna(value = claim[i].mode().iloc[0], inplace = True)
# claim
```

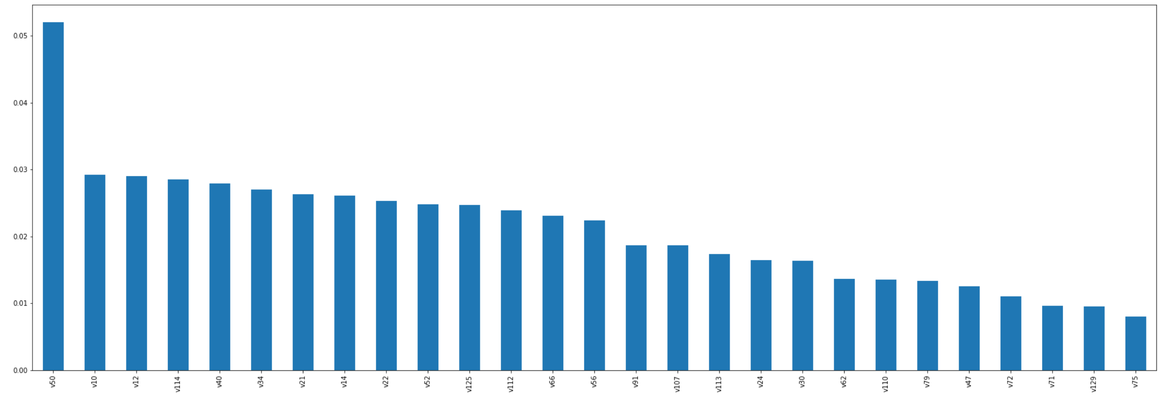
In [9]:

```
from sklearn.feature_selection import SelectKBest, f_classif
X = claim.drop(['ID', 'target'], axis = 1)
y = claim['target']
bestFeature = SelectKBest(score_func = f_classif, k = 20).fit(X, y)
be_features = pd.Series(bestFeature.scores_,X.columns)
best_features = be_features.nlargest(20)
plt.figure(figsize = (20, 10))
best_features.plot(kind = 'bar', color = 'r')
plt.show()
```



In [41]:

```
from sklearn.ensemble import ExtraTreesClassifier
feature = ExtraTreesClassifier().fit(X, y)
feat_importance = pd.Series(feature.feature_importances_, X.columns).nlargest(27)
plt.figure(figsize = (30, 10))
feat_importance.plot(kind = 'bar')
plt.show()
```

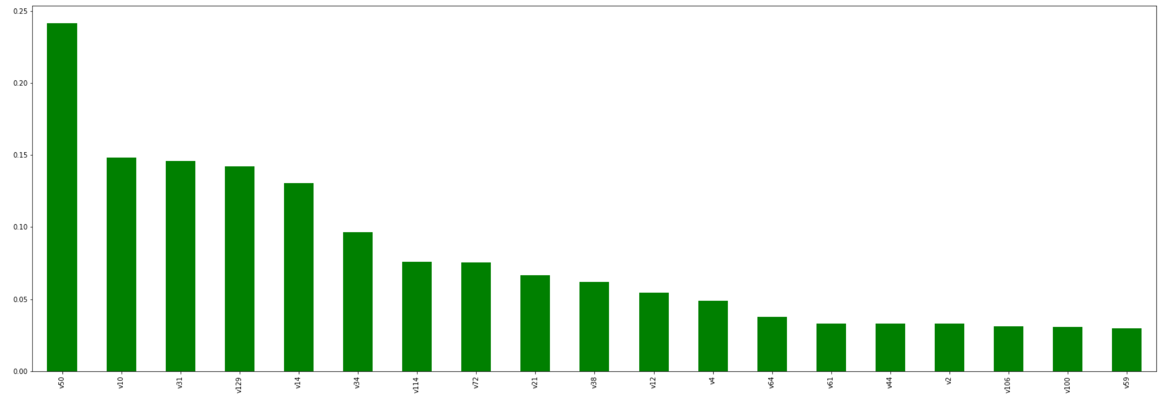



In [12]:

```
cormax = claim.corr()
#top_features = cormax.index
# plt.figure(figsize = (200, 200))
# sns.heatmap(cormax, annot=True, cmap="RdYlGn")
# plt.show()
```

In [14]:

```
best_corr_features = cormax['target'].sort_values(ascending = False)[:20][1:]
plt.figure(figsize = (30, 10))
best_corr_features.plot.bar(color = 'g')
plt.show()
```



In [15]:

```
most_imp_features = []
for i in X.columns:
    if i in feat_importance or i in best_corr_features or i in best_features:
        most_imp_features.append(i)
most_imp_features
```

Out[15]:

```
['v2',
 'v4',
 'v10',
 'v12',
 'v14',
 'v21',
 'v22',
 'v24',
 'v30',
 'v31',
 'v34',
 'v38',
 'v40',
 'v44',
 'v47',
 'v48',
 'v50',
 'v52',
 'v56',
 'v59',
 'v61',
 'v62',
 'v64',
 'v66',
 'v71',
 'v72',
 'v75',
 'v79',
 'v91',
 'v100',
 'v106',
 'v107',
 'v110',
 'v112',
 'v113',
 'v114',
 'v119',
 'v123',
 'v125',
 'v129']
```

In [16]:

```
X_imp = claim[most_imp_features]
X_imp
```

Out[16]:

| | v2 | v4 | v10 | v12 | v14 | v21 | v22 | v24 | v30 | v31 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 8.727474 | 3.921026 | 0.503281 | 6.085711 | 11.636387 | 7.730923 | 0 | 0 | 0 | 0 |
| 1 | 11.506664 | 4.915740 | 1.312910 | 6.507647 | 11.636386 | 6.763110 | 1 | 0 | 0 | 0 |
| 2 | 5.310079 | 4.410969 | 0.765864 | 6.384670 | 9.603542 | 5.245035 | 2 | 1 | 1 | 0 |
| 3 | 8.304757 | 4.225930 | 6.542669 | 9.646653 | 14.094723 | 7.517125 | 3 | 2 | 0 | 1 |
| 4 | 11.506664 | 4.915740 | 1.050328 | 6.320087 | 10.991098 | 6.414567 | 4 | 1 | 1 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 114316 | 11.506664 | 4.915740 | 1.444201 | 6.368061 | 11.865255 | 7.088172 | 6905 | 1 | 1 | 0 |
| 114317 | 11.506664 | 4.915740 | 6.236324 | 9.443324 | 14.924483 | 8.455263 | 2371 | 2 | 1 | 1 |
| 114318 | 11.506664 | 4.915740 | 2.078775 | 6.698925 | 12.269012 | 6.570625 | 7435 | 3 | 0 | 1 |
| 114319 | 11.506664 | 4.915740 | 1.291029 | 6.692204 | 12.573678 | 7.730751 | 9742 | 2 | 2 | 0 |
| 114320 | 7.932978 | 4.640085 | 0.853391 | 6.306396 | 11.967826 | 7.496000 | 70 | 0 | 0 | 0 |

114321 rows × 40 columns

In [17]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_imp, y, test_size = 0.2, random_s
tate = 0)
```

In [42]:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression().fit(X_train, y_train)
lr_train_score = lr.score(X_train, y_train)
print('Accuracy on train data', lr_train_score)
lr_score = lr.score(X_test, y_test)
print('Accuracy on test data', lr_score)
```

```
Accuracy on train data 0.7726229006298111
Accuracy on test data 0.7752460091843428
```

In [19]:

```
from sklearn.metrics import confusion_matrix
confusion = confusion_matrix(y_test, lr.predict(X_test))
print('Logistic regression confusion metrics\n', confusion)
```

```
Logistic regression confusion metrics
 [[  524  4928]
 [  211 17202]]
```

In [20]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, lr.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.71      0.10      0.17      5452
           1       0.78      0.99      0.87     17413

    accuracy                           0.78     22865
   macro avg       0.75      0.54      0.52     22865
weighted avg       0.76      0.78      0.70     22865
```

In [21]:

```
y_score_lr = lr.decision_function(X_test)
lr_score_list = list(zip(y_test[0:20], y_score_lr[0:20]))
lr_score_list
```

Out[21]:

```
[(1, -0.36129358961326147),
 (0, 1.2250526716794847),
 (1, 0.5718062255583997),
 (0, 0.13000089332807993),
 (1, 1.439069767345494),
 (1, 0.8241956413054017),
 (1, 0.3476949908467173),
 (0, 0.7144136215053187),
 (0, 0.3707391389158977),
 (1, 0.5897840617186382),
 (0, 0.06855227792471873),
 (1, 1.301339982104703),
 (0, 0.4359688031467399),
 (1, 1.3976182956196244),
 (0, 2.171151502825489 7),
 (1, 0.7474904079625183),
 (1, 2.3219650923800867),
 (1, 1.4967916400265795),
 (0, -0.03575264845202199),
 (0, 1.135489654792013)]
```

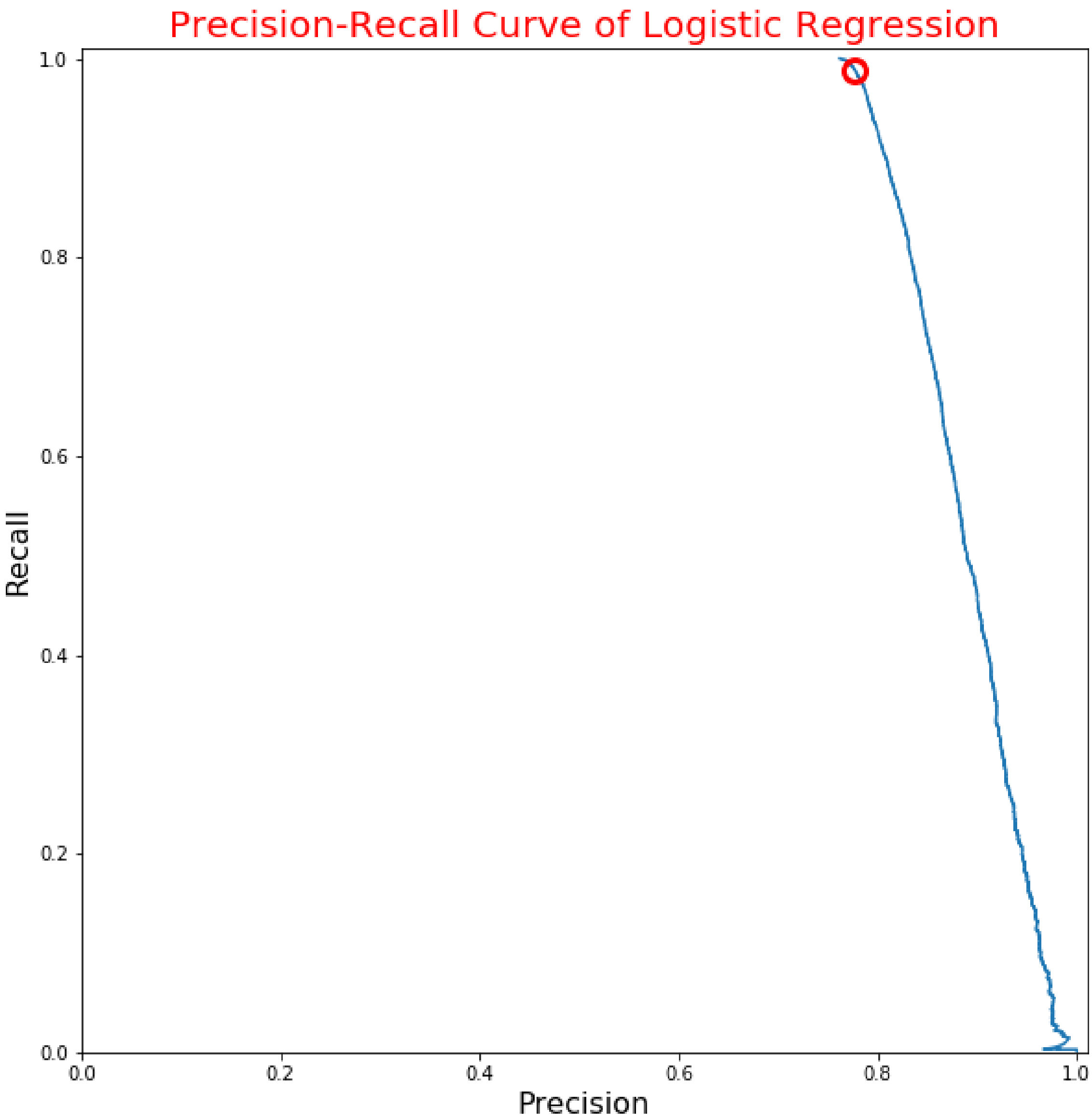
In [22]:

```
from sklearn.metrics import average_precision_score
average_precision = average_precision_score(y_test, y_score_lr)

print('Average precision-recall score: {0:0.2f}'.format(
      average_precision))
```

```
Average precision-recall score: 0.89
```

In [43]:

```
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds = precision_recall_curve(y_test, y_score_lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure(figsize = (10, 10))
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle = 'none', c=
'r', mew=3)
plt.title('Precision-Recall Curve of Logistic Regression', fontsize = 20, c = 'r')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```


Precision-Recall Curve of Logistic Regression
Recall
Precision
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

In [45]:

```
from sklearn.svm import LinearSVC
svc = LinearSVC().fit(X_train, y_train)
print('Accuracy on train data', svc.score(X_train, y_train))
print('Accuracy on test data', svc.score(X_test, y_test))
```

```
Accuracy on train data 0.7613059832050385
Accuracy on test data 0.7615132298272469
```

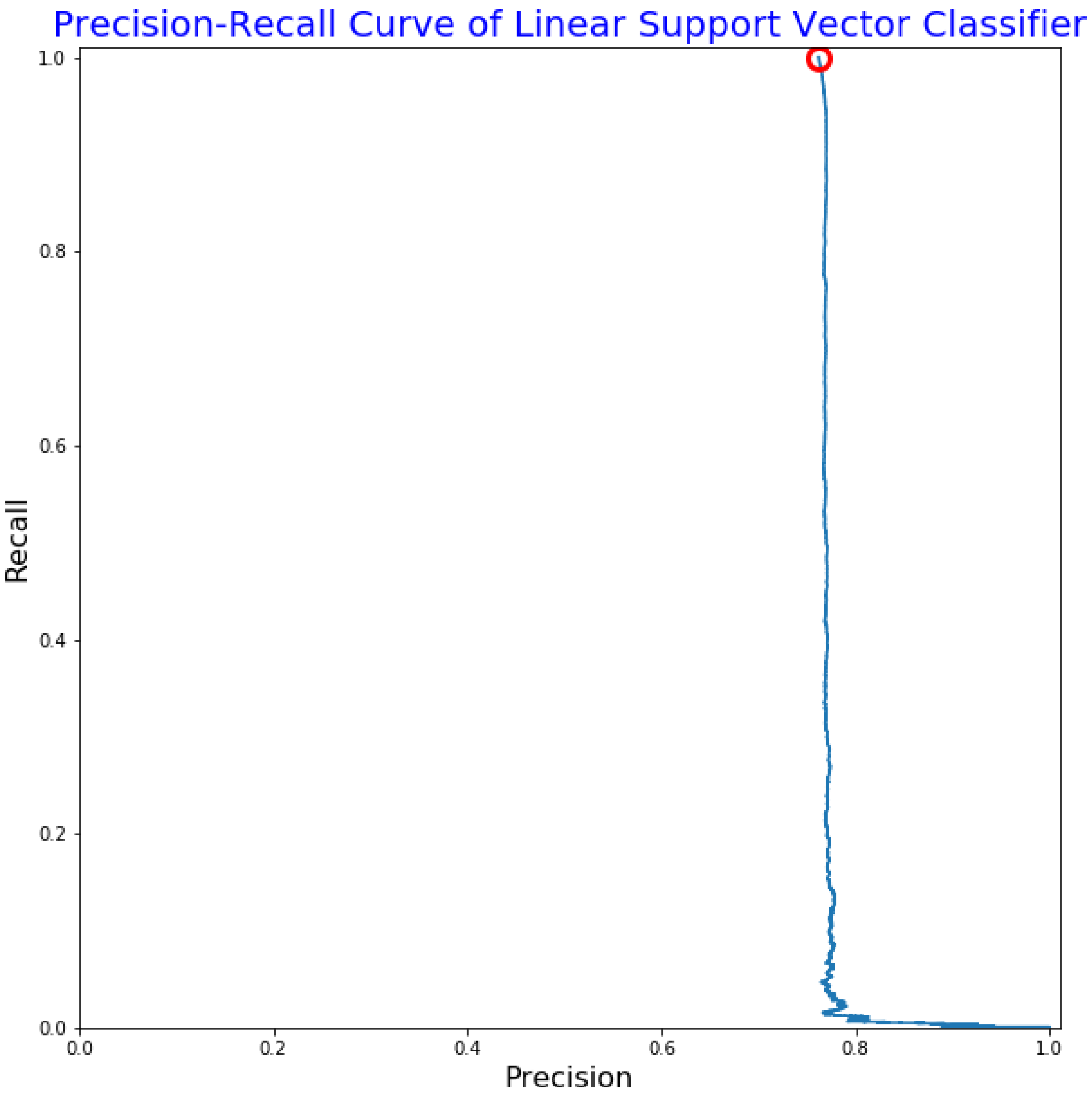
In [25]:

```
from sklearn.metrics import average_precision_score
y_score_svc = svc.decision_function(X_test)

average_precision = average_precision_score(y_test, y_score_svc)

print('Average precision-recall score: {0:0.2f}'.format(
      average_precision))
```

```
Average precision-recall score: 0.77
```

In [47]:

```
precision, recall, thresholds = precision_recall_curve(y_test, y_score_svc)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure(figsize = (10, 10))
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle = 'none', c=
'r', mew=3)
plt.title('Precision-Recall Curve of Linear Support Vector Classifier', fontsize = 20,
c = 'b')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```




In [48]:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier().fit(X_train, y_train)
print('Accuracy on train data', rfc.score(X_train, y_train))
print('Accuracy on test data', rfc.score(X_test, y_test))
```

```
Accuracy on train data 0.9924225857242828
Accuracy on test data 0.7509293680297398
```

In [35]:

```
confusion = confusion_matrix(y_test, lr.predict(X_test))
print('Random Forest Classifier confusion metrics\n', confusion)
```

```
Random Forest Classifier confusion metrics
 [[  524  4928]
 [  211 17202]]
```

In [36]:

```
print(classification_report(y_test, rfc.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.47      0.33      0.39      5452
           1       0.81      0.88      0.84     17413

    accuracy                           0.75     22865
   macro avg       0.64      0.61      0.62     22865
weighted avg       0.73      0.75      0.74     22865
```

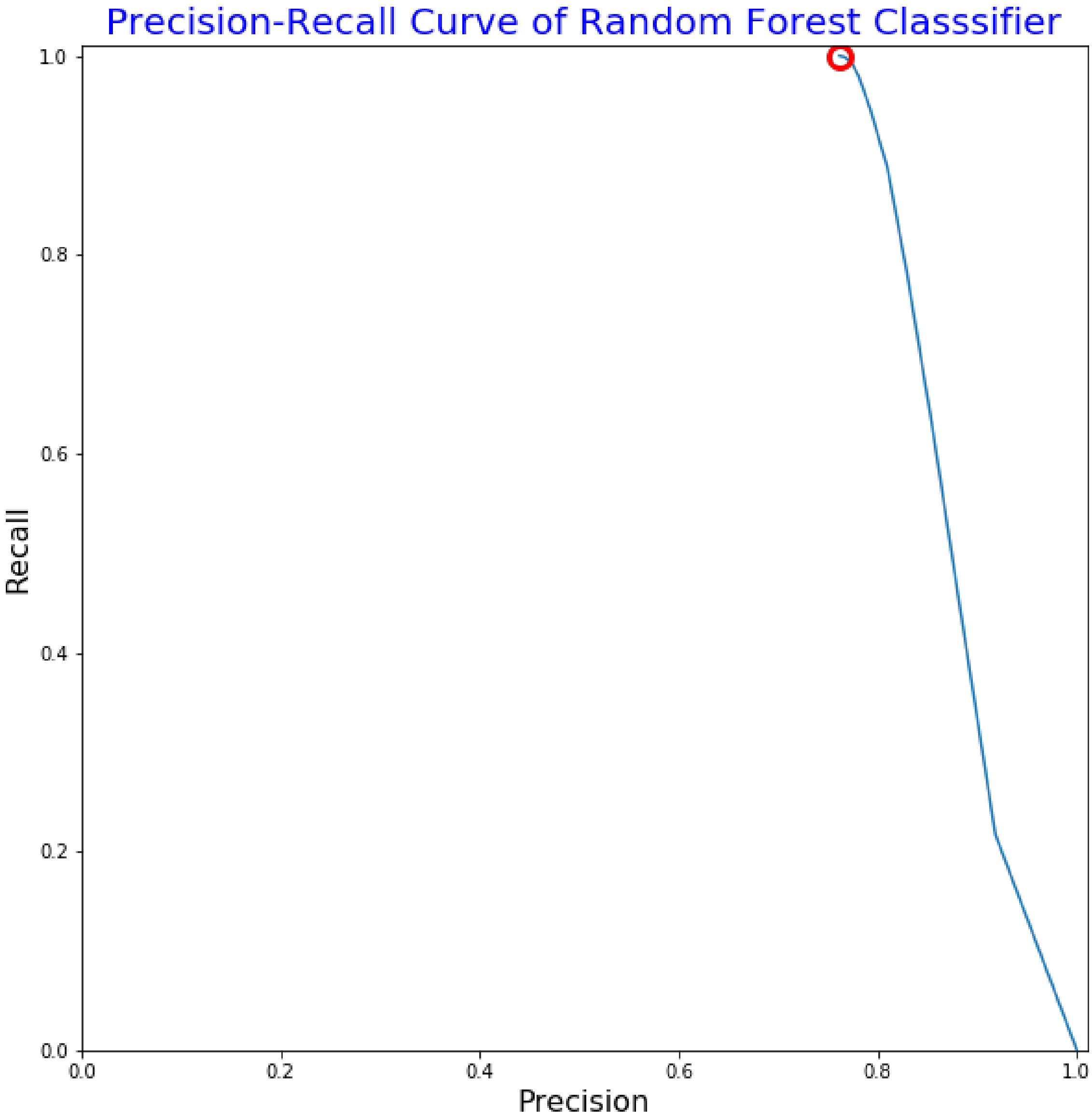
In [28]:

```
y_score_rfc = rfc.predict_proba(X_test)[:,-1]
print('Average precision-recall score RF: {}'.format(average_precision_score(y_test, y_
score_rfc)))
```

```
Average precision-recall score RF: 0.8583035825631781
```

In [55]:

```
precision, recall, thresholds = precision_recall_curve(y_test, y_score_rfc)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure(figsize = (10, 10))
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle = 'none', c=
'r', mew=3)
plt.title('Precision-Recall Curve of Random Forest Classsifier', fontsize = 20, c = 'b'
)
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```


Precision-Recall Curve of Random Forest Classsifier
Recall
Precision
0.0
0.2
0.4
0.6
0.8
1.0

In [ ]: