Case Study 5:
Firewall Traffic Classification

Kebur Fantahun, Eli Kravez, Halle Purdom

March 14, 2022

Introduction

For this report a company wants to develop an auto classifier that can determine whether to accept, deny, or drop their incoming firewall requests. Currently this process is done using a lot of manpower and resources, so the goal is to develop a classifier that is both accurate and efficient. Historical data from the company is supplied in order to create the models. In this project, support vector machines and stochastic gradient descent models are used and optimized to create classification models that can allow, deny, or drop firewall requests for the company.

Methods

Data Preparation

The company supplied 65,532 rows of historical data with 11 features and 1 target variable. The 'action' target variable initially had four categories: allow, deny, drop, and reset both. For the purposes of this study, the reset-both category was dropped as directed since it only had 54 entries, and the models were built for multiclass classification between the three remaining categories. Looking more into the distribution of the target variable, about 57% of the requests were allowed, 23% were denied, and 20% were dropped. The target was converted to integer values of 0 for allow, 1 for drop, and 2 for deny. There were no missing values in the data, although there were a significant amount of 0 values in a few of the features. All the features that were ports including Source Port, Destination Port, NAT Source Port, and NAT Destination port were converted to categorical features, since the ports represent addresses and are not ordinal values. Before feeding data into the models, it is split into categorical and numerical features. The numerical features are passed through the sklearn preprocessing StandardScaler.

Model Development

The data was shuffled and divided into 80% training set and 20% testing set. The training data was used to train and optimize the models, and the testing data was used to evaluate their performance.

Linear Support Vector Machine Classification (SVC) Model Development

A SVC model was created using the training data. To find the best parameters for this model, RandomizedGridSearch was used with 3 fold cross validation. The parameters tuned for this model included the following:

Parameter	Values
SVC_tol	np.arange(0.001,100,3)
SVC__loss	['hinge', 'squared_hinge']
SVC__C	[0.001, 0.0001, 0.01]

As seen above, the SVC_tol was given a set of values ranging from 0 to 100 to test several tolerances. From the randomized grid search, the final model SVC_tol value was changed to 0.001. The best accuracy is found when there is a low tolerance. The SVC_loss was tuned between hinge and squared hinge to see which may provide better results. The final SVC model is discussed below in Results.

Stochastic Gradient Descent Classification (SGD C) Model Development

A SGD C model was also created using the training data. To find the best parameters for this model, the below parameters were tested to find the optimal combination. The randomized grid search used in SVC was not needed since SGD more efficiently runs on the larger dataset with less runtime. The parameters tuned for this model included the following:

Parameter	Values
alpha	[0.0001, 0.001, 0.01, 0.1]
penalty	["l2", "l1"]
tol	[1e-3, 1e-4, 1e-2]

Alpha was given a set of values ranging from 0.0001 to 0.1 to find the best constant to multiply the regularization term by. The parameter search yielded 0.0001 as the best alpha. The best accuracy is found when the regularization term constant multiple is lowest and therefore makes a weaker regularization. The penalty was tuned between L1 and L2 regularization to see which penalty assists with accuracy best. The stopping criterion parameter 'tol' was also tuned between three values of 1e-3, 1e-4, and 1e-2. The final SGD C model is discussed below in Results.

Dataset Challenges

Because of the large size of the original dataset, there were challenges in modeling the data with both SVR and SGD. The dataset is 65,532 rows, and originally had 11 features. Because there were four port columns, which are categorical, the one hot encoding of these variables increased the feature amount to 57,628 features. The large size of the dataset led to very long runtimes in addition to memory issues of the computer used. In this study, the code was run on a computing system with high ram, so results were able to be obtained. However, this same code did not run on computers with smaller ram because of the memory issue, as confirmed when reducing the dataset size eliminated the error.

Methods explored include reducing integer variables from 64 to 8 bit, however since there are only 7 integer variables this made little to no effect on the memory size of the dataframe. The integer variables were also scaled with the StanderScaler to reduce runtime of the models. The Vowpal Wabbit out-of-core method for SGD C was tried, however the one-hot encoding of categorical variables also made this method too slow.

Results

After optimizing model parameters for SVC, the final SVC model uses linear SVC with the below parameters:

LinearSVC(max_iter = 10000, C = 66.01, tol = 0.001, loss = 'hinge')

After optimizing model parameters for SGD C, the final SGD C model can be seen below:

SGDClassifier(loss = 'log', alpha = 0.0001, penalty = 'l1', max_iter = 1000, tol= 0.001)

The two models are compared below. As seen in the table the SGD C model outperforms the SVC model in terms of precision, recall, and accuracy.

Model	Precision	Recall	Accuracy
SVC	0.99150	.99122	0.99121
SGD C	0.99787	0.99786	0.99786

Conclusion

In conclusion, the SGD C model produced higher precision, recall and accuracy than the SVC model. Although SVM methods have scaling issues with larger datasets, SGD is expected to perform better because of the lack of scaling issue and ability to tune the model parameters more easily and efficiently due to the decreased runtime. In the future, a more exhaustive grid search would provide better parameter tuning for both models to increase accuracy but would increase time cost.

In [ ]:

```
import pandas as pd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, confusion_matrix, acc
from sklearn.model_selection import train_test_split, cross_val_score, Stratifie
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split, cross_val_score, Stratifie
```

In [ ]:

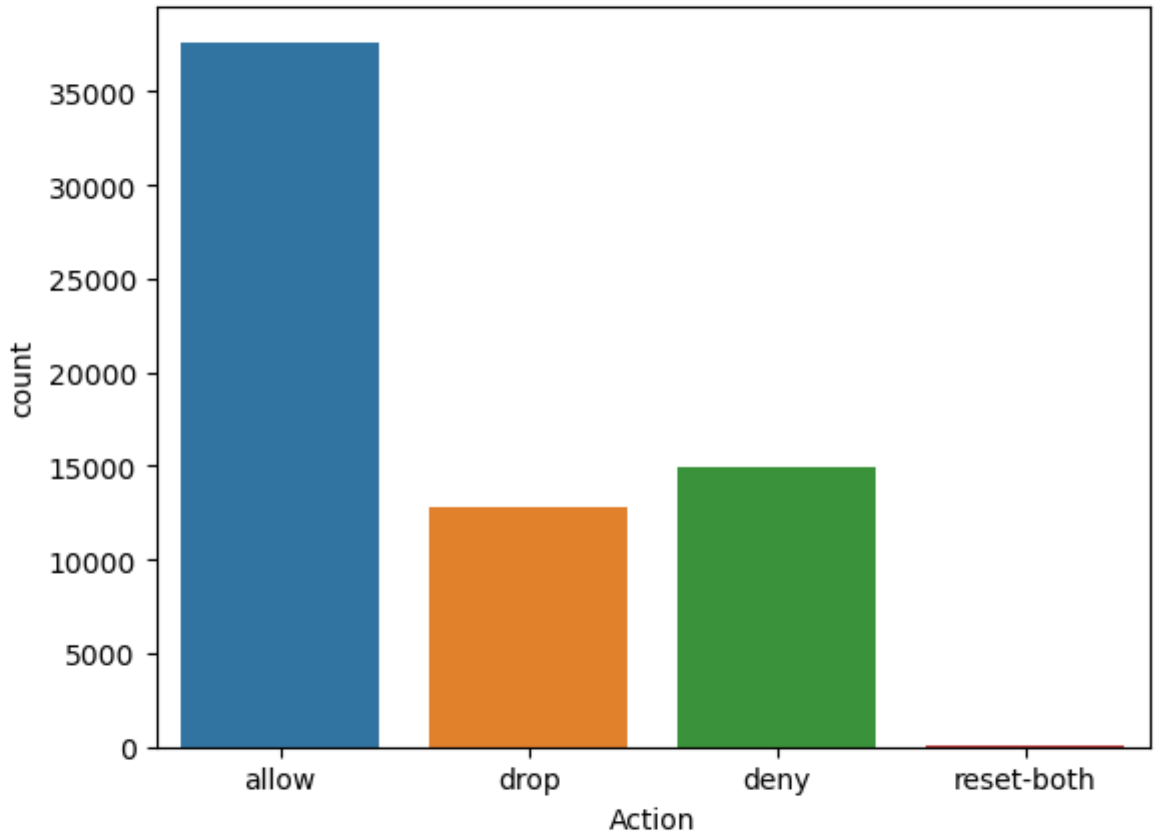
```
# read data
log2_df = pd.read_csv('log2.csv')
log2_df.head()
```

Out[ ]:

	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Action	Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (sec)
0	57222	53	54587	53	allow	177	94	83	2	30
1	56258	3389	56258	3389	allow	4768	1600	3168	19	17
2	6881	50321	43265	50321	allow	238	118	120	2	1199
3	50553	3389	50553	3389	allow	3327	1438	1889	15	17
4	50002	443	45848	443	allow	25358	6778	18580	31	16

In [ ]:

```
# class vizualization - we need to drop reset - both
ax = sns.countplot(x='Action', data=log2_df)
plt.show()
```



In [ ]:
```
# drop reset-both
log2_df = log2_df[log2_df['Action'].str.contains("reset-both") == False]
```

In [ ]:
```
# switch Action to be numerical
#log2_df['Action'] = pd.Categorical(log2_df.Action).codes #  pd.Categorical(log2

y = pd.factorize(log2_df['Action'])
log2_df['Action'] = y[0]
print(log2_df.dtypes)
```

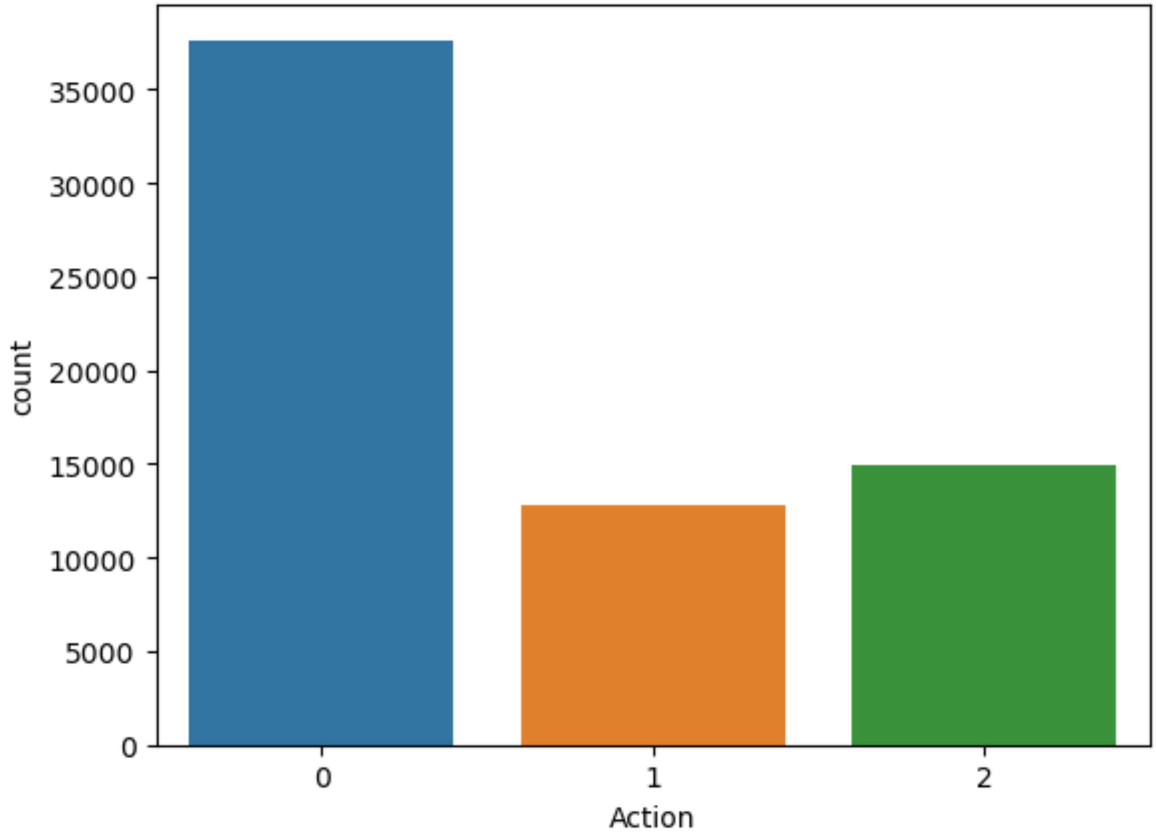
```
Source Port              int64
Destination Port         int64
NAT Source Port          int64
NAT Destination Port     int64
Action                   int64
Bytes                    int64
Bytes Sent               int64
Bytes Received           int64
Packets                  int64
Elapsed Time (sec)       int64
pkts_sent                int64
pkts_received            int64
dtype: object
```

In [ ]:
```
print(y)
```

```
(array([0, 0, 0, ..., 1, 1, 1], dtype=int64), Index(['allow', 'drop', 'deny'], d
type='object'))
```

In [ ]:
```
ax = sns.countplot(x='Action', data=log2_df)
```

```
plt.show()
```



In [ ]:

```
columns_categorical = ['Source Port', 'Destination Port', 'NAT Source Port', 'NA

log2_df_str =log2_df
# categoricals as string
log2_df_str[columns_categorical] = log2_df_str[columns_categorical].astype(str)

#target var
target = log2_df_str['Action']

features_categorica = log2_df_str[columns_categorical]

#numerical features
features_numerical = log2_df_str[['Bytes', 'Bytes Sent',        'Bytes Received'
```

In [ ]:

```
# categoricals as dummy
features_categorica_encode = pd.get_dummies(features_categorica,prefix=columns_c
```

In [ ]:

```
scaler = StandardScaler()
```

In [ ]:

```
#standard scale for numerical
df_scaled = pd.DataFrame(scaler.fit_transform(features_numerical),columns = feat
print(df_scaled.head())
```

```
      Bytes  Bytes Sent  Bytes Received   Packets  Elapsed Time (sec)  \
0 -0.017262   -0.005826       -0.030321 -0.019659           -0.118607
```

```
1 -0.016446   -0.005432       -0.029069 -0.016348           -0.161571
2 -0.017252   -0.005819       -0.030306 -0.019659            3.744857
3 -0.016702   -0.005475       -0.029588 -0.017127           -0.161571
4 -0.012782   -0.004080       -0.022815 -0.014011           -0.164876

   pkts_sent  pkts_received
0  -0.012556      -0.027208
1  -0.009761      -0.023611
2  -0.012556      -0.027208
3  -0.010382      -0.024510
4  -0.008829      -0.019565
```

In [ ]:

```
print(df_scaled.shape)
print()
print(features_categorica_encode.shape)
```

```
(65478, 7)

(65478, 57628)
```

In [ ]:

```
# joined features set
features = df_scaled.join(features_categorica_encode)

features.head()
```

Out[ ]:

	Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (sec)	pkts_sent	pkts_received	Source Port_10000
0	-0.017262	-0.005826	-0.030321	-0.019659	-0.118607	-0.012556	-0.027208	0
1	-0.016446	-0.005432	-0.029069	-0.016348	-0.161571	-0.009761	-0.023611	0
2	-0.017252	-0.005819	-0.030306	-0.019659	3.744857	-0.012556	-0.027208	0
3	-0.016702	-0.005475	-0.029588	-0.017127	-0.161571	-0.010382	-0.024510	0
4	-0.012782	-0.004080	-0.022815	-0.014011	-0.164876	-0.008829	-0.019565	0

5 rows × 57635 columns

In [ ]:

```
features[features.isna().any(axis=1)]
```

Out[ ]:

Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (sec)	pkts_sent	pkts_received	Source Port_10000	Source Port_10001

0 rows × 57635 columns

In [ ]:

```
features.shape
```

Out[ ]: (65478, 57635)

SGDClassifier

In [ ]:
```
# my_model = SGDClassifier(loss = 'log')
# my_model.fit(features, target)
# preds = my_model.predict(features)
# accuracy_score(target, preds)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=
```

In [ ]:
```
alpha = [0.0001, 0.001, 0.01, 0.1] # learning_rate
penalty= ["l2", "l1"]
tol = [1e-3, 1e-4, 1e-2] # stopping criterion

for i in range(0, len(alpha)):
    for j in range(0, len(penalty)):
        for k in range (0, len(tol)):
            my_model = SGDClassifier(loss = 'log', alpha = alpha[i], penalty=pen
            my_model.fit(X_train, y_train)

            # to do - use Test here
            preds = my_model.predict(X_test)

            print('-----------------------------------------------------------
            print('alpha: ' , alpha[i], ' penalty: ', penalty[j], ' tol: ', tol[
            print('accuracy_score : ', accuracy_score(y_test, preds), ' precisio
```

```
--------------------------------------------------------------------------------
------
alpha:  0.0001  penalty:  l2  tol:  0.001
accuracy_score :  0.9881643249847282  precision_score :  0.9885019211354557  rec
all_score:  0.9881643249847282
--------------------------------------------------------------------------------
------
alpha:  0.0001  penalty:  l2  tol:  0.0001
accuracy_score :  0.988317043372022  precision_score :  0.988624278996042  recal
l_score:  0.988317043372022
--------------------------------------------------------------------------------
------
alpha:  0.0001  penalty:  l2  tol:  0.01
accuracy_score :  0.9882406841783751  precision_score :  0.9885824424274949  rec
all_score:  0.9882406841783751
--------------------------------------------------------------------------------
------
alpha:  0.0001  penalty:  l1  tol:  0.001
accuracy_score :  0.9978619425778864  precision_score :  0.997872018992245  reca
ll_score:  0.9978619425778864
--------------------------------------------------------------------------------
------
alpha:  0.0001  penalty:  l1  tol:  0.0001
accuracy_score :  0.9934331093463653  precision_score :  0.9935696591867296  rec
all_score:  0.9934331093463653
--------------------------------------------------------------------------------
------
alpha:  0.0001  penalty:  l1  tol:  0.01
accuracy_score :  0.9977092241905925  precision_score :  0.997717778493645  reca
ll_score:  0.9977092241905925
--------------------------------------------------------------------------------
------
alpha:  0.001  penalty:  l2  tol:  0.001
accuracy_score :  0.946090409285278  precision_score :  0.9558436429335337  reca
ll_score:  0.946090409285278
--------------------------------------------------------------------------------
```

```
------
alpha:  0.001  penalty:  l2  tol:  0.0001
accuracy_score :  0.9458613317043372  precision_score :  0.9555569022129183  rec
all_score:  0.9458613317043372
--------------------------------------------------------------------------------
------
alpha:  0.001  penalty:  l2  tol:  0.01
accuracy_score :  0.946090409285278  precision_score :  0.9558436429335337  reca
ll_score:  0.946090409285278
--------------------------------------------------------------------------------
------
alpha:  0.001  penalty:  l1  tol:  0.001
accuracy_score :  0.9474648747709224  precision_score :  0.9567569710128777  rec
all_score:  0.9474648747709224
--------------------------------------------------------------------------------
------
alpha:  0.001  penalty:  l1  tol:  0.0001
accuracy_score :  0.9479230299328039  precision_score :  0.9570634223542592  rec
all_score:  0.9479230299328039
--------------------------------------------------------------------------------
------
alpha:  0.001  penalty:  l1  tol:  0.01
accuracy_score :  0.9486102626756261  precision_score :  0.9575249996762073  rec
all_score:  0.9486102626756261
--------------------------------------------------------------------------------
------
alpha:  0.01  penalty:  l2  tol:  0.001
accuracy_score :  0.9319639584605987  precision_score :  0.9469519261483744  rec
all_score:  0.9319639584605987
--------------------------------------------------------------------------------
------
alpha:  0.01  penalty:  l2  tol:  0.0001
accuracy_score :  0.9340256566890653  precision_score :  0.9481959811562701  rec
all_score:  0.9340256566890653
--------------------------------------------------------------------------------
------
alpha:  0.01  penalty:  l2  tol:  0.01
accuracy_score :  0.9340256566890653  precision_score :  0.9481959811562701  rec
all_score:  0.9340256566890653
--------------------------------------------------------------------------------
------
alpha:  0.01  penalty:  l1  tol:  0.001
accuracy_score :  0.9187538179596824  precision_score :  0.9393770009358117  rec
all_score:  0.9187538179596824
--------------------------------------------------------------------------------
------
alpha:  0.01  penalty:  l1  tol:  0.0001
accuracy_score :  0.9183720219914477  precision_score :  0.93916774313508  recal
l_score:  0.9183720219914477
--------------------------------------------------------------------------------
------
alpha:  0.01  penalty:  l1  tol:  0.01
accuracy_score :  0.9183720219914477  precision_score :  0.93916774313508  recal
l_score:  0.9183720219914477
--------------------------------------------------------------------------------
------
alpha:  0.1  penalty:  l2  tol:  0.001
accuracy_score :  0.7729841172877214  precision_score :  0.6096243164215274  rec
all_score:  0.7729841172877214
C:\Users\elikra\AppData\Local\Continuum\anaconda3\envs\ML7331\lib\site-packages
\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision is i
ll-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
--------------------------------------------------------------------------------
```

```
------
alpha:  0.1  penalty:  l2  tol:  0.0001
accuracy_score :  0.7729841172877214  precision_score :  0.6096243164215274  rec
all_score:  0.7729841172877214
```

```
C:\Users\elikra\AppData\Local\Continuum\anaconda3\envs\ML7331\lib\site-packages
\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision is i
ll-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
--------------------------------------------------------------------------------
------
alpha:  0.1  penalty:  l2  tol:  0.01
accuracy_score :  0.7729841172877214  precision_score :  0.6096243164215274  rec
all_score:  0.7729841172877214
```

```
C:\Users\elikra\AppData\Local\Continuum\anaconda3\envs\ML7331\lib\site-packages
\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision is i
ll-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
--------------------------------------------------------------------------------
------
alpha:  0.1  penalty:  l1  tol:  0.001
accuracy_score :  0.5797953573610263  precision_score :  0.3361626564174001  rec
all_score:  0.5797953573610263
```

```
C:\Users\elikra\AppData\Local\Continuum\anaconda3\envs\ML7331\lib\site-packages
\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision is i
ll-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
--------------------------------------------------------------------------------
------
alpha:  0.1  penalty:  l1  tol:  0.0001
accuracy_score :  0.5797953573610263  precision_score :  0.3361626564174001  rec
all_score:  0.5797953573610263
```

```
C:\Users\elikra\AppData\Local\Continuum\anaconda3\envs\ML7331\lib\site-packages
\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision is i
ll-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
--------------------------------------------------------------------------------
------
alpha:  0.1  penalty:  l1  tol:  0.01
accuracy_score :  0.5797953573610263  precision_score :  0.3361626564174001  rec
all_score:  0.5797953573610263
```

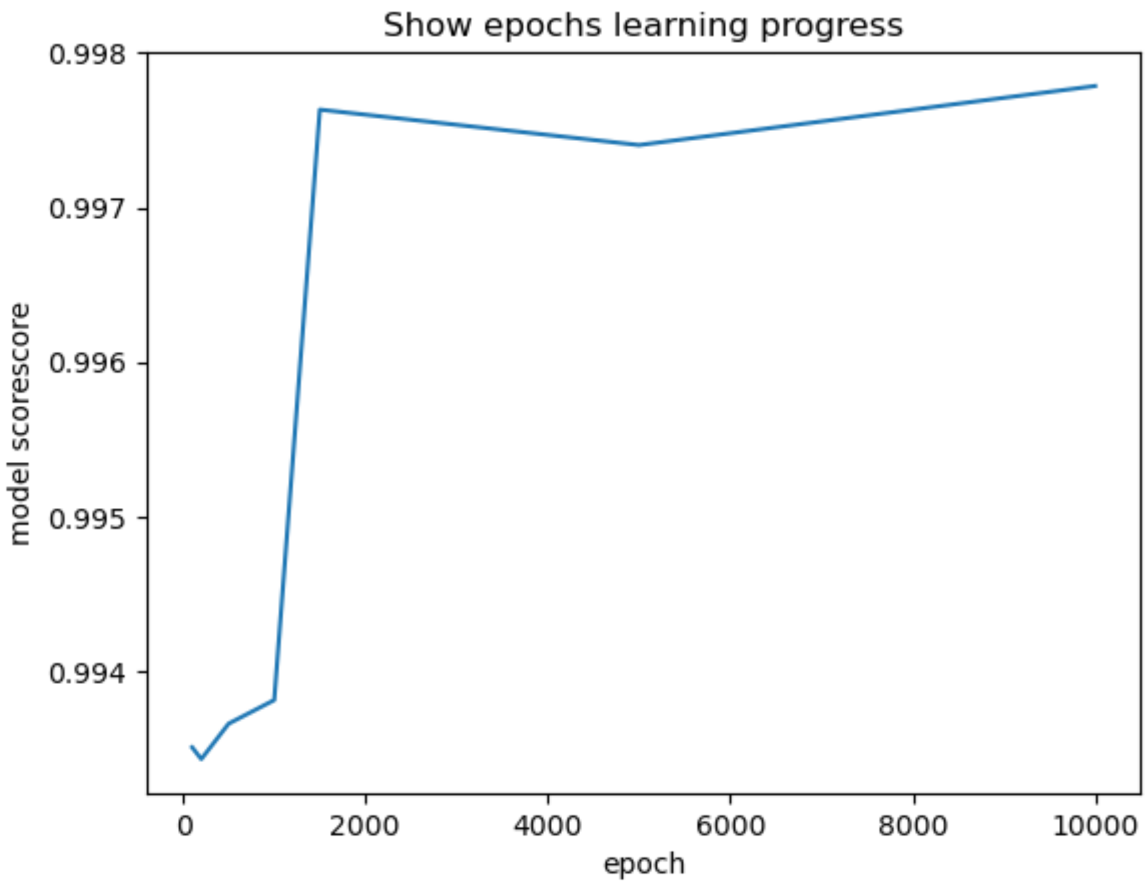
```
C:\Users\elikra\AppData\Local\Continuum\anaconda3\envs\ML7331\lib\site-packages
\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision is i
ll-defined and being set to 0.0 in labels with no predicted samples. Use `zero_d
ivision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]:

```
# add loop like with best params to see learning process :

epochs = [100, 200, 500, 1000, 1500, 5000, 10000] # epochs
model_scores = []
model_scores_train = []

for epoch in epochs:

    model = SGDClassifier(loss="log", penalty="l1", max_iter=epoch, alpha = 0.00
    model.fit(X_train, y_train)
    model_scores.append(model.score(X_test, y_test))
    model_scores_train.append(model.score(X_train, y_train))
```

In [ ]:

```
plt.title("Show epochs learning progress")
plt.xlabel("epoch")
plt.ylabel("model scorescore")
plt.plot(epochs, model_scores)
plt.show()
```



LinearSVC

In [ ]:

```
SVCpipe = Pipeline([#('scale', StandardScaler()),
                   ('SVC',LinearSVC(max_iter = 10000))])

# check StandardScaler?
param_grid = {'SVC__C':np.arange(0.01,100,3),
              'SVC__tol': [1e-3, 1e-4, 1e-2],    # stopping criterion
              'SVC__loss': ['hinge', 'squared_hinge'] } # np.arange(0.01,100,20
linearSVC = RandomizedSearchCV(SVCpipe,param_grid,return_train_score=True, n_job
```

In [ ]:

```
np.arange(0.001,100,3)
```

Out[ ]:

```
array([1.0000e-03, 3.0010e+00, 6.0010e+00, 9.0010e+00, 1.2001e+01,
       1.5001e+01, 1.8001e+01, 2.1001e+01, 2.4001e+01, 2.7001e+01,
       3.0001e+01, 3.3001e+01, 3.6001e+01, 3.9001e+01, 4.2001e+01,
       4.5001e+01, 4.8001e+01, 5.1001e+01, 5.4001e+01, 5.7001e+01,
       6.0001e+01, 6.3001e+01, 6.6001e+01, 6.9001e+01, 7.2001e+01,
```

```
       7.5001e+01, 7.8001e+01, 8.1001e+01, 8.4001e+01, 8.7001e+01,
       9.0001e+01, 9.3001e+01, 9.6001e+01, 9.9001e+01])
```

In [ ]:

```
linearSVC.fit(X_train,y_train)
print(linearSVC.best_params_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
C:\Users\elikra\AppData\Local\Continuum\anaconda3\envs\ML7331\lib\site-packages
\sklearn\model_selection\_search.py:925: UserWarning: One or more of the test sc
ores are non-finite: [nan nan nan nan nan nan nan nan nan nan]
  category=UserWarning
C:\Users\elikra\AppData\Local\Continuum\anaconda3\envs\ML7331\lib\site-packages
\sklearn\model_selection\_search.py:925: UserWarning: One or more of the train s
cores are non-finite: [nan nan nan nan nan nan nan nan nan nan]
  category=UserWarning
{'SVC__tol': 0.01, 'SVC__loss': 'hinge', 'SVC__C': 6.01}
```

In [ ]:

```
print(linearSVC.best_params_)
```

```
{'SVC__tol': 0.01, 'SVC__loss': 'hinge', 'SVC__C': 6.01}
```

In [ ]:

```
model_LinearSVC = LinearSVC(max_iter = 10000, C = 66.01, tol = 0.001, loss = 'hi
```

In [ ]:

```
model_LinearSVC.fit(X_train,y_train)
```

Out[ ]:

```
LinearSVC(C=66.01, loss='hinge', max_iter=10000, tol=0.001)
```

In [ ]:

```
model_LinearSVC.fit(X_train, y_train)
preds_svc = model_LinearSVC.predict(X_test)

print('accuracy_score : ', accuracy_score(y_test, preds_svc), ' precision_score
```

```
accuracy_score :  0.9912186927306048  precision_score :  0.99149811707031  recal
l_score:  0.9912186927306048
```

In [ ]: