NAME: AVIREDDY NVSRK ROHAN

REG.NO: 19BCE1180

LOGISTIC REGRESSION

DATASET SOURCE:https://www.kaggle.com/nsaravana/malware-detection

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
data=pd.read_csv("/content/Malware dataset.csv")
data.head(5)
```

	hash	millisecond	classification	state
0	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	0	malware	0
1	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	1	malware	0
2	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	2	malware	0
3	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	3	malware	0
4	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	4	malware	0

data description and splitting

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 35 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   hash               100000 non-null  object
 1   millisecond        100000 non-null  int64 
 2   classification     100000 non-null  object
 3   state              100000 non-null  int64 
 4   usage_counter      100000 non-null  int64 
 5   prio               100000 non-null  int64 
 6   static_prio        100000 non-null  int64 
 7   normal_prio        100000 non-null  int64 
 8   policy             100000 non-null  int64 
 9   vm_pgoff           100000 non-null  int64 
 10  vm_truncate_count  100000 non-null  int64 
 11  task_size          100000 non-null  int64 
 12  cached_hole_size   100000 non-null  int64 
 13  free_area_cache    100000 non-null  int64 
 14  mm_users           100000 non-null  int64 
 15  map_count          100000 non-null  int64 
 16  hiwater_rss        100000 non-null  int64 
 17  total_vm           100000 non-null  int64 
 18  shared_vm          100000 non-null  int64 
 19  exec_vm            100000 non-null  int64 
 20  reserved_vm        100000 non-null  int64 
 21  nr_ptes            100000 non-null  int64 
 22  end_data           100000 non-null  int64 
 23  last_interval      100000 non-null  int64 
 24  nvcsw              100000 non-null  int64 
 25  nivcsw             100000 non-null  int64 
 26  min_flt            100000 non-null  int64 
 27  maj_flt            100000 non-null  int64 
 28  fs_excl_counter    100000 non-null  int64 
 29  lock               100000 non-null  int64 
 30  utime              100000 non-null  int64 
 31  stime              100000 non-null  int64 
 32  gtime              100000 non-null  int64 
 33  cgtime             100000 non-null  int64 
 34  signal_nvcsw       100000 non-null  int64 
dtypes: int64(33), object(2)
memory usage: 26.7+ MB
```

```
data.describe()
```

	millisecond	state	usage_counter	prio	static_prio	normal_p
count	100000.000000	1.000000e+05	100000.0	1.000000e+05	100000.000000	1000

```
for i in data.columns:

    print(i+":" +str(data[i].isna().sum()))
```

```
hash:0
millisecond:0
classification:0
state:0
usage_counter:0
prio:0
static_prio:0
normal_prio:0
policy:0
vm_pgoff:0
vm_truncate_count:0
task_size:0
cached_hole_size:0
free_area_cache:0
mm_users:0
map_count:0
hiwater_rss:0
total_vm:0
shared_vm:0
exec_vm:0
reserved_vm:0
nr_ptes:0
end_data:0
last_interval:0
nvcsw:0
nivcsw:0
min_flt:0
maj_flt:0
fs_excl_counter:0
lock:0
utime:0
stime:0
gtime:0
cgtime:0
signal_nvcsw:0
```

```
for i in data.columns:
  print(i)
  print(data[i].unique())
  print(" ")
```

```
hash
['42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0'
 'com.kmcpesh.medicalskillsproceduresfree.apk' 'com.roidapp.photogrid.apk'
 '5dd6c684ad85ec01c32172a38451f0b6f3b261dce3c335dbe099d87763fe7790'
 'biz.mtoy.blockpuzzle.revolution.apk'
```

```
       '35e61d9b00a30f757d9b96fe5e5e2c89a8ebaa6e5787eb4b7e3a4a4213c4ce97'
       'air.com.freshplanet.games.MoviePop.apk' 'com.imangi.templerun2.apk'
       'com.kvadgroup.photostudio.apk' 'com.vbsmojivy.mianzed.apk'
       '3c722b32535e6e8ea1bbf8accdff73376cf2c3393d9325304ffc37c3213fdb4c'
       'com.modernenglishstudio.HowToSpeak.apk'
       'com.king.candycrushsodasaga.apk' 'com.google.zxing.client.android.apk'
       'com.tyengl.vocab.apk' 'imoblife.toolbox.full.apk'
       'com.tangram3D.AthleticsFree.apk' 'com.venticake.retrica.apk'
       'com.microsoft.amp.apps.bingfinance.apk'
       'com.androiddevelopermx.blogspot.organos3d.apk'
       'com.microsoft.office.word.apk' 'com.ezmusicplayer.demo.apk'
       'com.piriform.ccleaner.apk' 'com.ea.game.tetris2011_na.apk'
       'com.miniclip.dudeperfect.apk' 'audio.mp3.music.player.apk'
       'com.baiwang.instablend.apk' 'Bible_apkpure.com.apk' 'com.qizz.life.apk'
       'com.i6.FlightSimulatorAirplane3D.apk' 'example.matharithmetics.apk'
       'air.com.KalromSystems.BestButtFitness.apk'
       'com.mobilityware.solitaire.apk' 'air.com.tensquaregames.letsfish.apk'
       '156a617d84b92c1611e153ebaa1fc2e9d1af9c6154834c20d1f414c4d61e1983'
       '024b27972a6b3a1535510e9c0f154fb1a8e3a2afb25d5c30d2f6a9d23424d925'
       '186d3233e77f4a0c64043da385fb7f0dcd195ee0c1f46d3e8f49d4bf8d5d2d1f'
       'com.fitnesskeeper.runkeeper.pro.apk' 'com.bti.myGuitar.apk'
       'com.jrtstudio.music.apk' 'com.gotv.nflgamecenter.us.lite.apk'
       '1dec265aeda7b58e4173f47af0641a949937edbf21904ff1b6681c5348642387'
       'com.baiwang.PhotoFeeling.apk' 'com.medicaljoyworks.prognosis.apk'
       'com.fingerprintplay.bysbaseball2015.apk'
       '2f7693ee9f8a349d6c8f4e1a90a9c6a41b774d48c17aad205d3dfc799a4d74a3'
       '4872481a573f7c048db06b467bb68405febe870a45916d01e21b1e1216ea294f'
       'com.google.android.apps.docs.editors.docs.apk'
       '797ca0705a3b8220e671660849c9ab8f030c09e8c00224a375a92802d521fab3'
       'com.pfinance.apk' 'com.rottzgames.realjigsaw.apk'
       'com.ludia.familyfeudblitz.apk' 'com.zynga.wordsontour.apk'
       'com.sonypicturestelevision.sportsjeopardy.apk'
       'com.appquiz.educational.games.apk' 'DOCECG2.doctor.apk'
       '4fd8fb06b479ae910df2e8806d3a6968a921c9cad596cc06aadd5c7e04c7df2f'
       '3d131647f203a5283ef2488c1d48c93f72b201d422675df552c62b7069a3bc2b'
       'com.epocrates.apk' 'com.figure1.android.apk'
       'com.zayaninfotech.english.grammar.apk' 'com.magmamobile.game.Words.apk'
       '1117d14765e9169184cc931f7a417a460898e4b0d8f3c86562065fc82f5866ce'
       '2e185a901298b5ec69b2b22538a1fec71a4434fa495ffb802fa3a5558c31c91c'
       'com.music.choice.apk'
       '30d6fb78a81325c38c8d4d48a43d4e9f5c0621436e781c76e8be7965b6c3c988'
       '32effc5a6bc3b7319b5b7da02a7cc3576d44c1794b335be71ab8f3545a0555bc'
       '2e4c54588cca3be3ae471b4b9b531ed2a70b2d336688fef6d5bbed7ba21db580'
       '7590e4a832b9a17bdea9904cc84d8d132ce8b218b7e7e40086509d5f6f728ac7'
       '7ea81b362027866c147218ffa657a1ccc59d677f540bb0cb5a98f7546f18ed82'
       '711415bfe471619f1dd4dbdae0a9d82f5c01b44f57501d59f1d37479411b50c4'
       '116ae92ecfacb70146fe643d92878e522f71af393702f3b66d2135a06bcff57f'
       '46203ffdacf94fed4a78011bcdfb7378ad36bd815a5ee8c9f1836ca590ccc075'
       '84892f7a0b371c835ad31d4462222646a610925873961084de2ebd8486d478ad'
       '6e222eabc2ef5f8077c379f3b4da0e77bd0bca71b7e0320df49831e749a2a568'
       '0602834d897fe3f3314586ae867aed63f3757be01b7f0354c8626519d8575453'
       '6f158980b71ae3d8ffc462fa6c63ca0a3cc0a7ab7cc079bede1279c4f67fae23'
       '3d51872172186d55238444384224ca46d0f1a7ab87910494c6354a7a53074387'
       '1824056efb105d20db233bfeb1f93ee69eeaff81b63eb8cd53d582d7330687ab'
```

```
total_target=data["classification"]
data_id=data["hash"]
data.drop("classification",inplace=True,axis=1)
data.drop("classification",inplace=True,axis=1)
```

```
#DROPPING BECAUSE THERE IS ONLY ONE UNIQUE VALUE AND ASSUMING THEY CANNOT CONVEY ANY INFORMAT
data.drop("usage_counter",inplace=True,axis=1)
data.drop("normal_prio",inplace=True,axis=1)
data.drop("policy",inplace=True,axis=1)
data.drop("vm_pgoff",inplace=True,axis=1)
data.drop("task_size",inplace=True,axis=1)
data.drop("cached_hole_size",inplace=True,axis=1)
data.drop("hiwater_rss",inplace=True,axis=1)
data.drop("nr_ptes",inplace=True,axis=1)
data.drop("cgtime",inplace=True,axis=1)
data.drop("signal_nvcsw",inplace=True,axis=1)
```

```
total_target.value_counts()
```

```
     benign     50000
     malware    50000
     Name: classification, dtype: int64
```

```
data.drop("hash",inplace=True,axis=1)
```

```
train_data,test_data,train_labels,test_labels=train_test_split(data,total_target,test_size=0.
```

```
train_labels.value_counts()
```

```
     benign     35031
     malware    34969
     Name: classification, dtype: int64
```

```
test_labels.value_counts()
```

```
     malware    15031
     benign     14969
     Name: classification, dtype: int64
```

```
train_data.head()
```

	millisecond	state	prio	static_prio	vm_truncate_count	free_area_cache

```
print(train_data.shape)
print(train_labels.shape)
```

```
(70000, 23)
(70000,)
```

```
test_data.head()
```

n	exec_vm	reserved_vm	end_data	last_interval	nvcsw	nivcsw	min_flt	maj_flt	fs_ex
)	124	275	120	3804	342616	101	1	120	
4	145	283	114	2	349207	19	0	114	
)	127	193	120	4322	347766	1	1	120	
4	166	387	114	4257	355538	29	1	114	
)	96	82	120	0	337902	2	1	120	

```
print(test_data.shape)
print(test_labels.shape)
```

```
(30000, 23)
(30000,)
```

CLASS SIZES

```
np.unique(train_labels,return_counts=True)
```

```
(array(['benign', 'malware'], dtype=object), array([35031, 34969]))
```

Preprocessing

```
data.head()
```

lisecond	state	prio	static_prio	vm_truncate_count	free_area_cache	mm_users
0	0	3069378560	14274	13173	24	724
1	0	3069378560	14274	13173	24	724

```
#ENCODING THE CATEGORICAL COLUMNS (ONLY TARGET LABEL)
lenc=LabelEncoder()
train_labels=lenc.fit_transform(train_labels)
test_labels=lenc.fit_transform(test_labels)
```

```
test_labels
```

```
     array([1, 1, 0, ..., 1, 1, 1])
```

```
sc = StandardScaler()
sc.fit(data)
train_std = sc.transform(train_data)
test_std = sc.transform(test_data)
```

Logistic regression model using sklearn

```
lr = LogisticRegression(C=100.0, random_state=1, solver='liblinear', multi_class='ovr')
lr.fit(train_std,train_labels)
```

```
     LogisticRegression(C=100.0, class_weight=None, dual=False, fit_intercept=True,
                        intercept_scaling=1, l1_ratio=None, max_iter=100,
                        multi_class='ovr', n_jobs=None, penalty='l2', random_state=1,
                        solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

```
predicted=lr.predict(test_std)
```

without dropping constant columns

```
print(classification_report(test_labels,predicted))
```

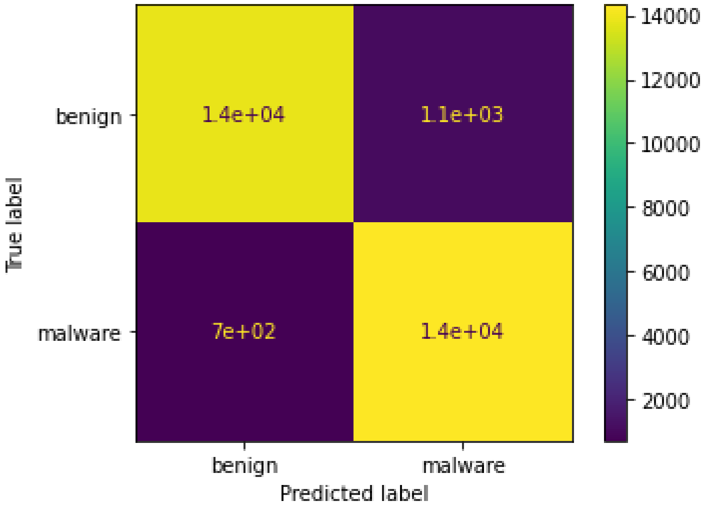
```
                   precision    recall  f1-score   support

          benign       0.95      0.93      0.94     14969
         malware       0.93      0.95      0.94     15031

        accuracy                           0.94     30000
       macro avg       0.94      0.94      0.94     30000
    weighted avg       0.94      0.94      0.94     30000
```

after dropping constant value columns

```
print(classification_report(test_labels,predicted))
```

```
              precision    recall  f1-score   support

           0       0.95      0.93      0.94     14969
           1       0.93      0.95      0.94     15031

    accuracy                           0.94     30000
   macro avg       0.94      0.94      0.94     30000
weighted avg       0.94      0.94      0.94     30000
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay


cm = confusion_matrix(test_labels, predicted)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=["benign","malware"])
disp.plot()
plt.show()
```



Double-click (or enter) to edit

```
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt


def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor='black')

    # highlight test examples
    if test_idx:
        # plot all examples
        X_test, y_test = X[test_idx, :], y[test_idx]

        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
                    c='',
                    edgecolor='black',
                    alpha=1.0,
                    linewidth=1,
                    marker='o',
                    s=100,
                    label='test set')


class LogisticRegressionGD(object):
    """Logistic Regression Classifier using gradient descent.

    Parameters
    ------------
    eta : float
      Learning rate (between 0.0 and 1.0)
    n_iter : int
      Passes over the training dataset.
    random_state : int
      Random number generator seed for random weight
      initialization.
```

```
Attributes
-----------
w_ : 1d-array
  Weights after fitting.
cost_ : list
  Logistic cost function value in each epoch.

"""
def __init__(self, eta=0.05, n_iter=100, random_state=1):
    self.eta = eta
    self.n_iter = n_iter
    self.random_state = random_state

def fit(self, X, y):
    """ Fit training data.

    Parameters
    ----------
    X : {array-like}, shape = [n_examples, n_features]
      Training vectors, where n_examples is the number of examples and
      n_features is the number of features.
    y : array-like, shape = [n_examples]
      Target values.

    Returns
    -------
    self : object

    """
    rgen = np.random.RandomState(self.random_state)
    self.w_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape[1])
    self.cost_ = []

    for i in range(self.n_iter):
        net_input = self.net_input(X)
        output = self.activation(net_input)
        errors = (y - output)
        self.w_[1:] += self.eta * X.T.dot(errors)
        self.w_[0] += self.eta * errors.sum()

        # note that we compute the logistic `cost` now
        # instead of the sum of squared errors cost
        cost = -y.dot(np.log(output)) - ((1 - y).dot(np.log(1 - output)))
        self.cost_.append(cost)
    return self

def net_input(self, X):
    """Calculate net input"""
    return np.dot(X, self.w_[1:]) + self.w_[0]

def activation(self, z):
    def activation(self, z):
        """Compute logistic sigmoid activation"""
        return 1. / (1. + np.exp(-np.clip(z, -250, 250)))

    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.net_input(X) >= 0.0, 1, 0)
        # equivalent to:
        # return np.where(self.activation(self.net_input(X)) >= 0.5, 1, 0)
```

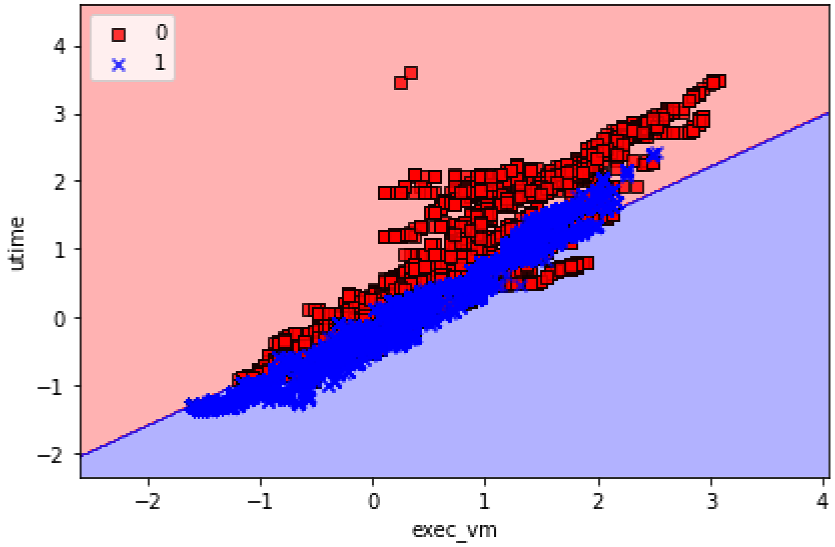
exec vm utime, utime min_users, mm users ncsvw, mmusers exec vm,mmusers vmtruncate count

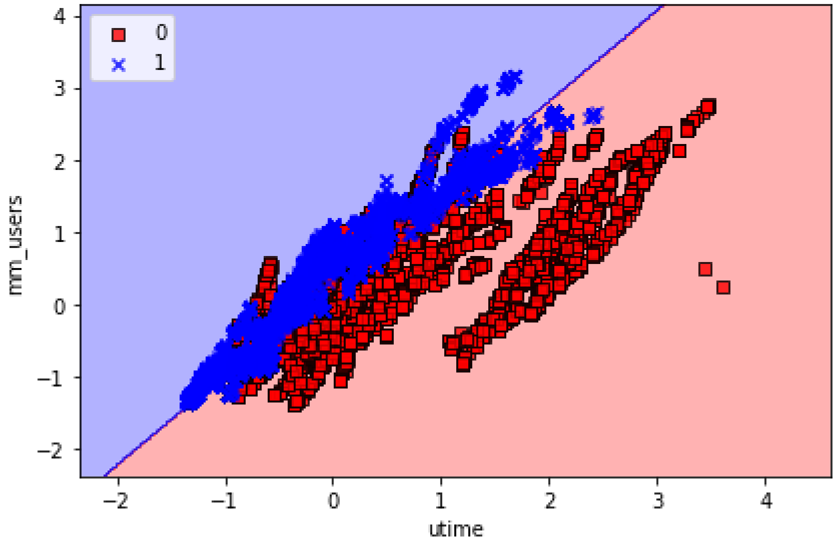
```
train_data.columns
```

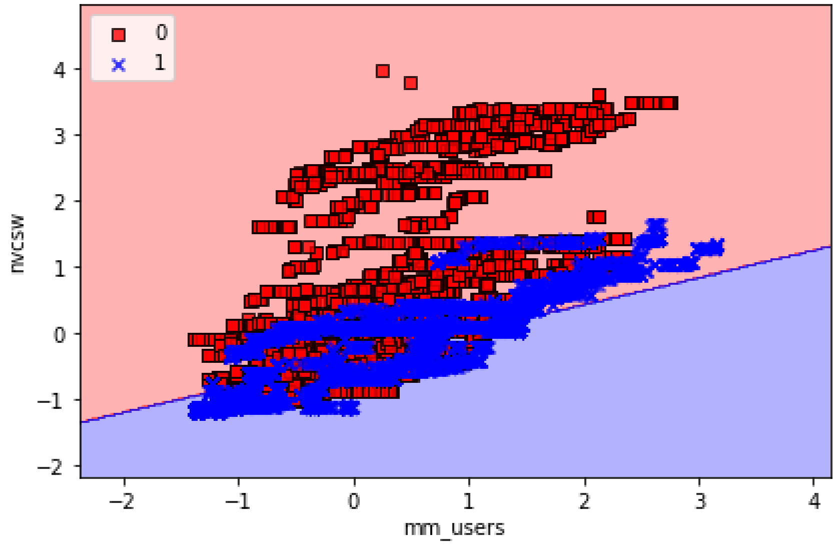
```
     Index(['millisecond', 'state', 'usage_counter', 'prio', 'static_prio',
            'normal_prio', 'policy', 'vm_pgoff', 'vm_truncate_count', 'task_size',
            'cached_hole_size', 'free_area_cache', 'mm_users', 'map_count',
            'hiwater_rss', 'total_vm', 'shared_vm', 'exec_vm', 'reserved_vm',
            'nr_ptes', 'end_data', 'last_interval', 'nvcsw', 'nivcsw', 'min_flt',
            'maj_flt', 'fs_excl_counter', 'lock', 'utime', 'stime', 'gtime',
            'cgtime', 'signal_nvcsw'],
           dtype='object')
```

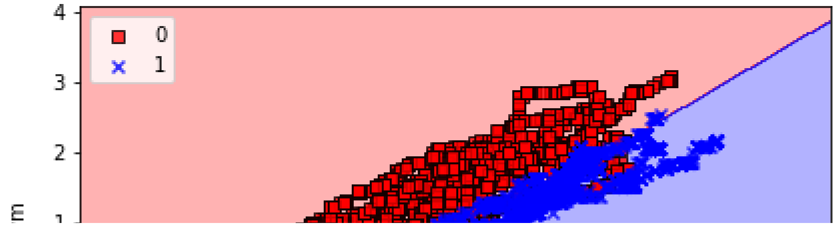
```
plot_list=[['exec_vm','utime'],['utime','mm_users'],['mm_users','nvcsw'],['mm_users','exec_vm
for·k·in·plot_list:
      train_subset = train_std[:,[train_data.columns.get_loc(k[0]),train_data.columns.get_loc

      lrgd = LogisticRegressionGD(eta=0.05, n_iter=1000, random_state=1)
      lrgd.fit(train_subset,train_labels)

      plot_decision_regions(X=train_subset, y=train_labels,classifier=lrgd)

      plt.xlabel(k[0])
      plt.ylabel(k[1])
      plt.legend(loc='upper left')

      plt.tight_layout()
      #plt.savefig('images/03_05.png', dpi=300)
      plt.show()
```

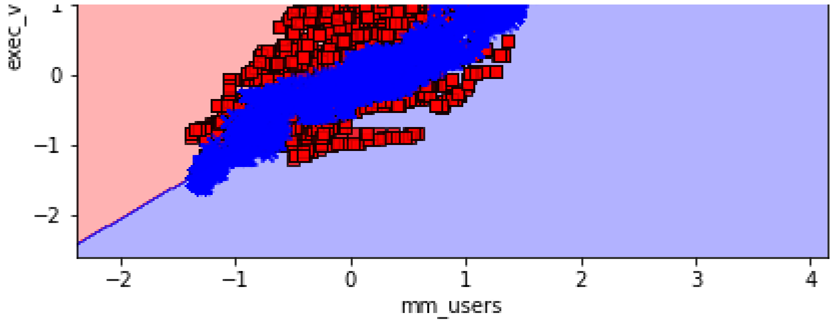
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:57: RuntimeWarning: divide
```

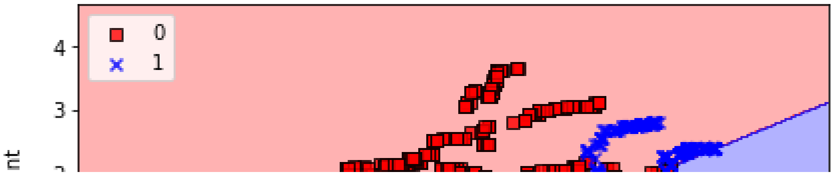












```
for k in plot_list:
  test_subset=test_std[:,[train_data.columns.get_loc(k[0]),train_data.columns.get_loc(k[1])]]
  lrgd.fit(train_subset,train_labels)
  print(k)
  print(classification_report(test_labels,lrgd.predict(test_subset)))
  print(" ")
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:57: RuntimeWarning: divide
['exec_vm', 'utime']
              precision    recall  f1-score   support

           0       0.78      0.55      0.64     14969
           1       0.65      0.85      0.74     15031

    accuracy                           0.70     30000
   macro avg       0.72      0.70      0.69     30000
weighted avg       0.72      0.70      0.69     30000


['utime', 'mm_users']
              precision    recall  f1-score   support

           0       0.38      0.33      0.35     14969
           1       0.41      0.46      0.43     15031

    accuracy                           0.40     30000
   macro avg       0.39      0.40      0.39     30000
weighted avg       0.39      0.40      0.39     30000


['mm_users', 'nvcsw']
              precision    recall  f1-score   support

           0       0.90      0.60      0.72     14969
           1       0.70      0.93      0.80     15031

    accuracy                           0.77     30000
   macro avg       0.80      0.76      0.76     30000
weighted avg       0.80      0.77      0.76     30000
```

```
['mm_users', 'exec_vm']
              precision    recall  f1-score   support

           0       0.72      0.55      0.62     14969
           1       0.64      0.79      0.71     15031

    accuracy                           0.67     30000
   macro avg       0.68      0.67      0.67     30000
weighted avg       0.68      0.67      0.67     30000


['mm_users', 'vm_truncate_count']
              precision    recall  f1-score   support

           0       0.77      0.61      0.68     14969
           1       0.68      0.82      0.74     15031

    accuracy                           0.71     30000
   macro avg       0.73      0.71      0.71     30000
weighted avg       0.73      0.71      0.71     30000
```

✓ 40s completed at 01:33 ● ×