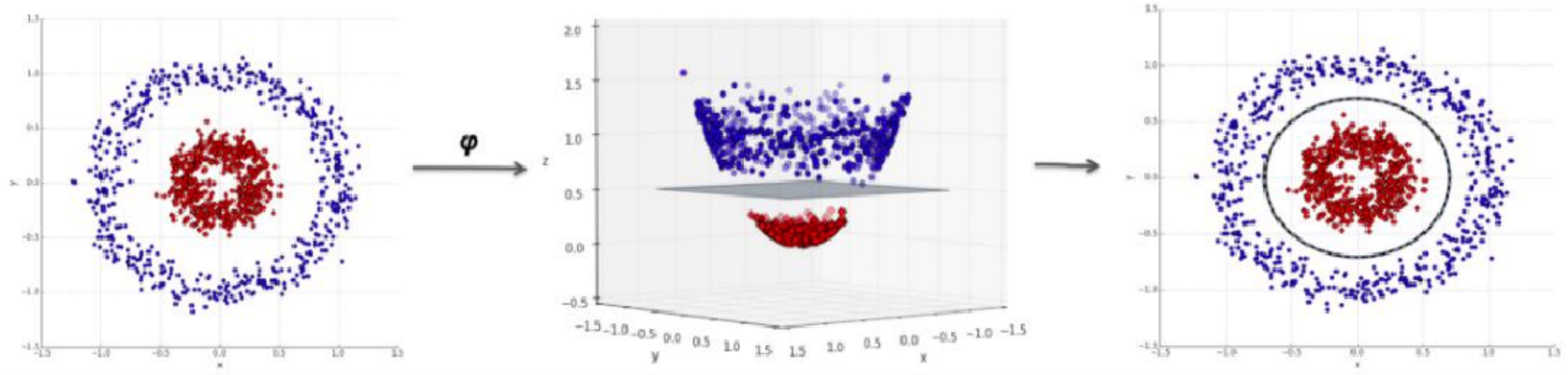
Practice 9

Kernel SVM

Introduction to Big Data Analytics(2020)

Problem

- ➢ **Use kernel SVM in Scikit-Learn library**
- ➢ **Predict whether each data point was extracted from facial skin image or not.**
  - ▪ Use predefined function in sklearn.**svm**



https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

https://www.semanticscholar.org/paper/Hypotheses-engine-(HypE)%3A-exploring-structured-in-Mano/e7e9715af59074624cb56fbfead533c5c5ab37cb/figure/6

Dataset

➢ Dataset description

· **The dataset is constructed over Blue, Green, Red color space.**

➢ 3 Features

1. Blue
2. Green
3. Red
4. Skin image(=1) or not(=2)

❖ **The last column of the dataset indicates the class labels.**

* UCI Machine Learning Repository :

http://archive.ics.uci.edu/ml/datasets/Skin+Segmentation

➢ **You can download the pre-processed train and test dataset on i-campus**

Practice 9

1. **Compare accuracy, F1 score and confusion matrix of linear SVM and kernel SVM.**

2. **Use predefined classes in *sklearn.svm.SVC***

   · **Configure "kernel" parameter of SVC class to set the type of SVM model.**

   · **Linear SVM : *kernel = "linear"***

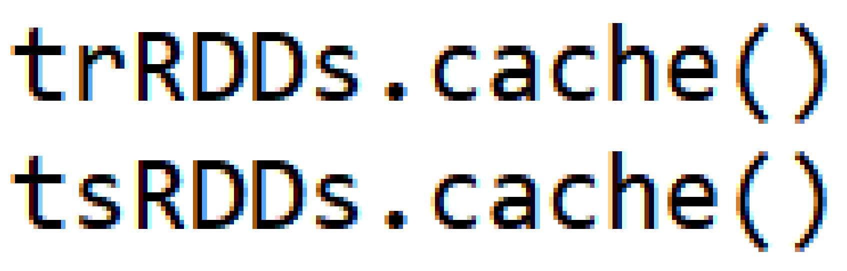
   · **Kernel SVM : *kernel = "rbf"***

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Practice 9

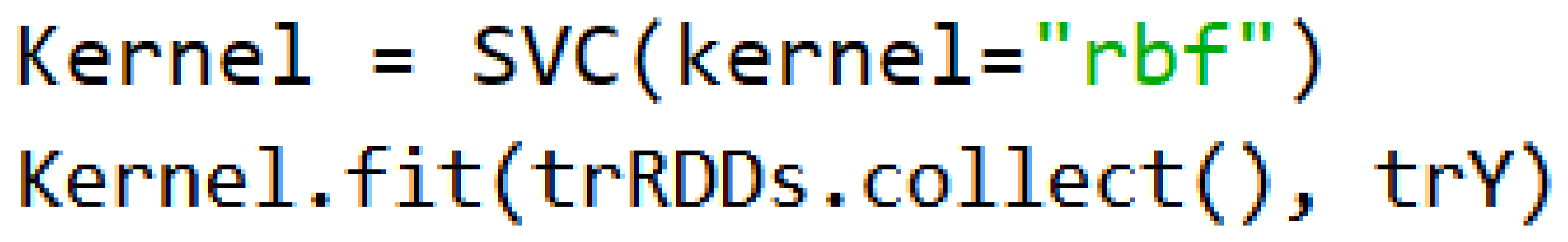
3. How to train the model using RDD data format

- Before training the model, you need to save data into your memory using ***cache()*** function.
- For example



```
trRDDs.cache()
tsRDDs.cache()
```

- In this example, ***trRDDs***: training data points(RDD) & ***tsRDDs***: test data points(RDD)
- Then, you can easily train SVM model provided by scikit-learn using ***fit()*** function
- For example



```
Kernel = SVC(kernel="rbf")
Kernel.fit(trRDDs.collect(), trY)
```

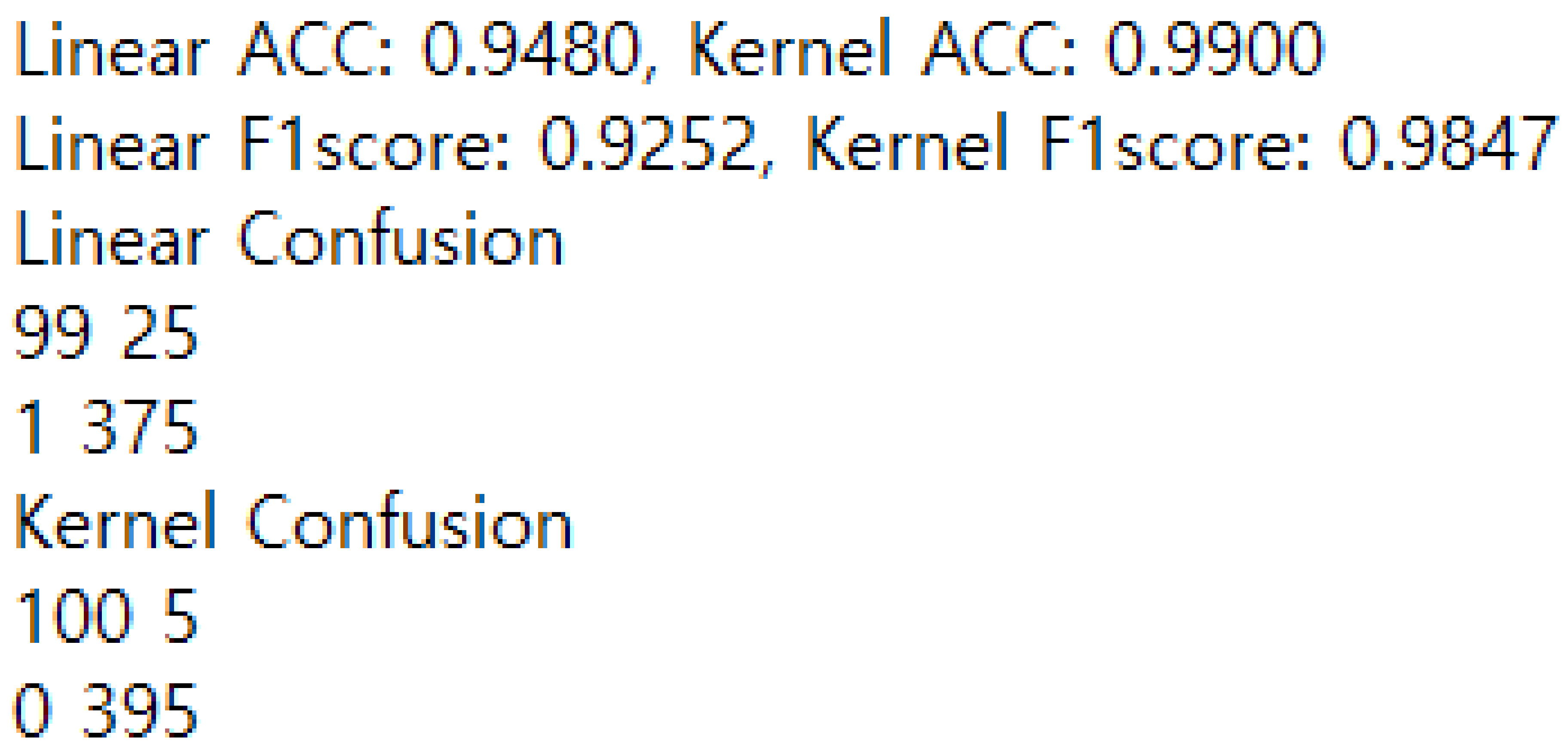
- In this example, ***trY***: training data points' label

4. Due date: June 11th 23:59

https://spark.apache.org/docs/latest/rdd-programming-guide.html
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

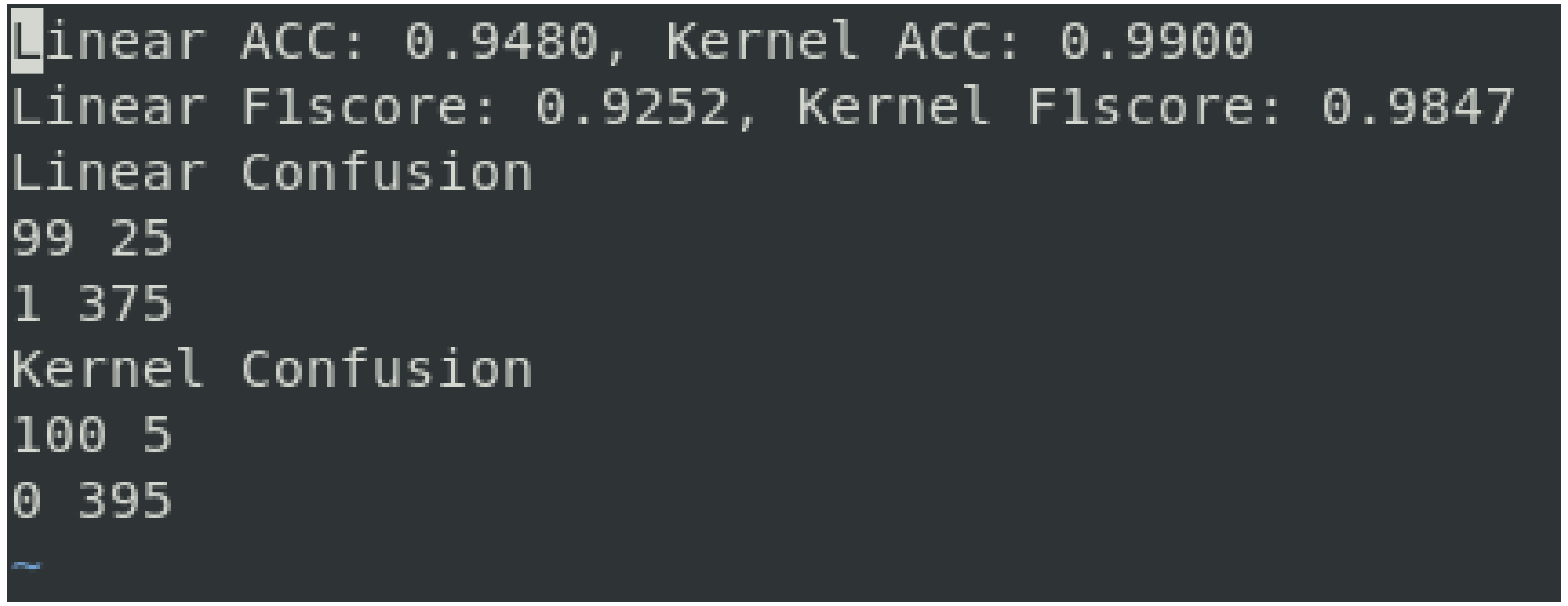
Submission

- ➢ **You need to submit *result.txt* file**
  - ✓ *Write **F1 score, accuracy** and **confusion matrix** of **linear SVM***
  - ✓ *Write **F1 score, accuracy** and **confusion matrix** of **kernel SVM***



```
Linear ACC: 0.9480, Kernel ACC: 0.9900
Linear F1score: 0.9252, Kernel F1score: 0.9847
Linear Confusion
99 25
1 375
Kernel Confusion
100 5
0 395
```

Windows

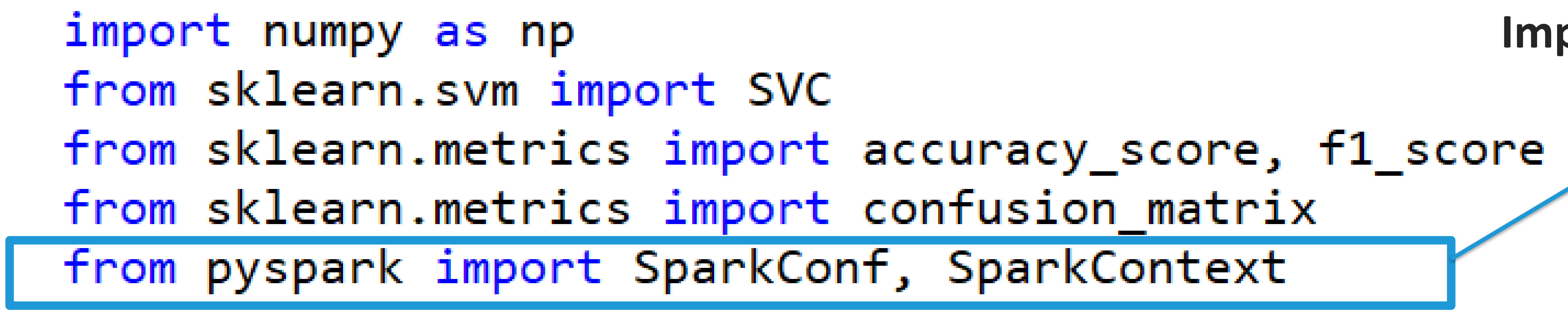


```
Linear ACC: 0.9480, Kernel ACC: 0.9900
Linear F1score: 0.9252, Kernel F1score: 0.9847
Linear Confusion
99 25
1 375
Kernel Confusion
100 5
0 395
~
```

Linux

Solution

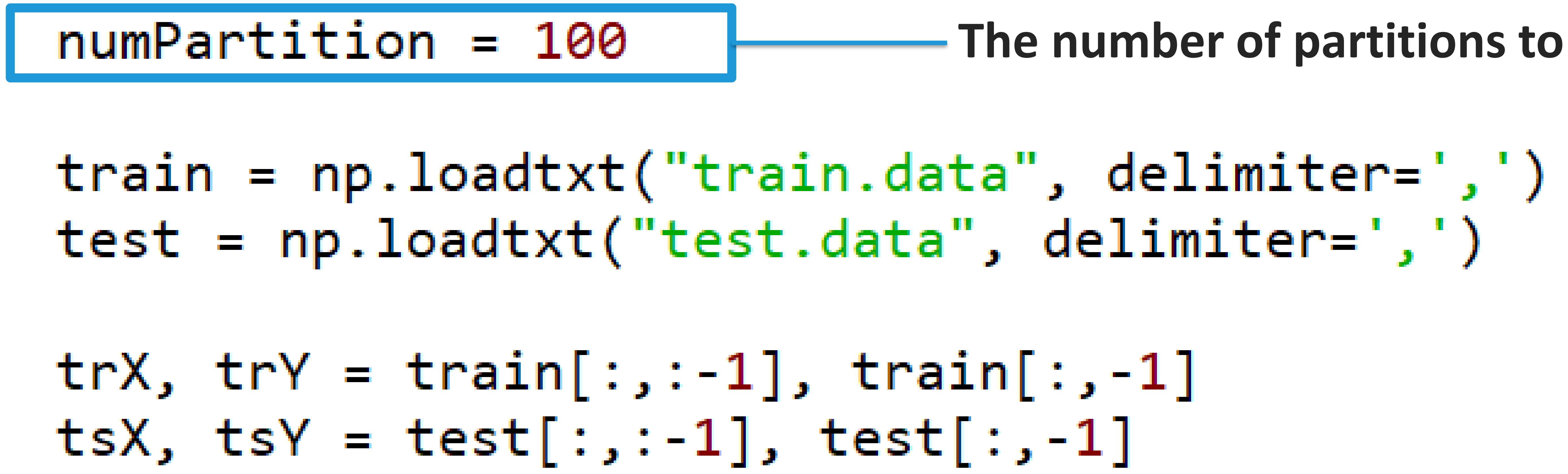
- ➢ Import package



```python
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import confusion_matrix
from pyspark import SparkConf, SparkContext
```

Import the Spark Package in your program

- ➢ Load train dataset and test dataset

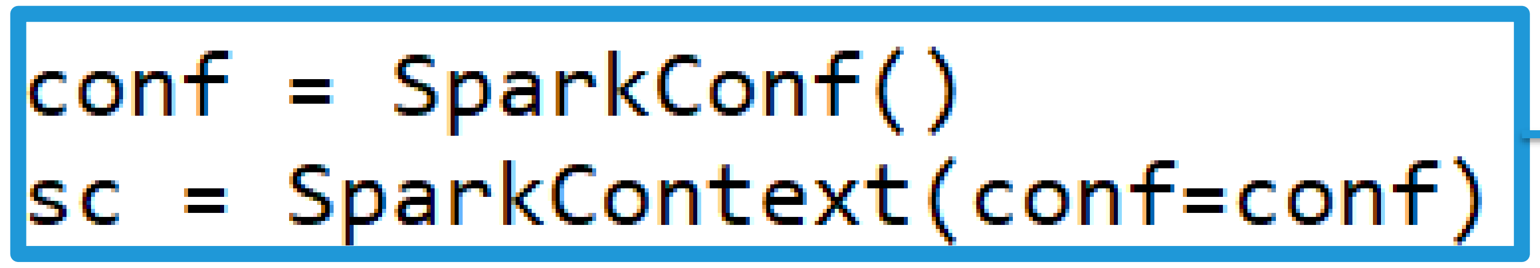


```python
numPartition = 100

train = np.loadtxt("train.data", delimiter=',')
test = np.loadtxt("test.data", delimiter=',')

trX, trY = train[:,:-1], train[:,-1]
tsX, tsY = test[:,:-1], test[:,-1]
```

The number of partitions to split the data

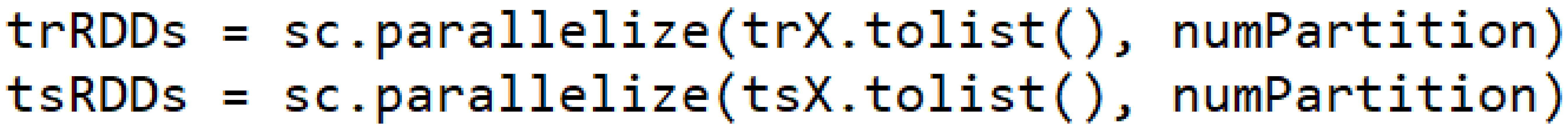
Solution

- Initialize a SparkContext

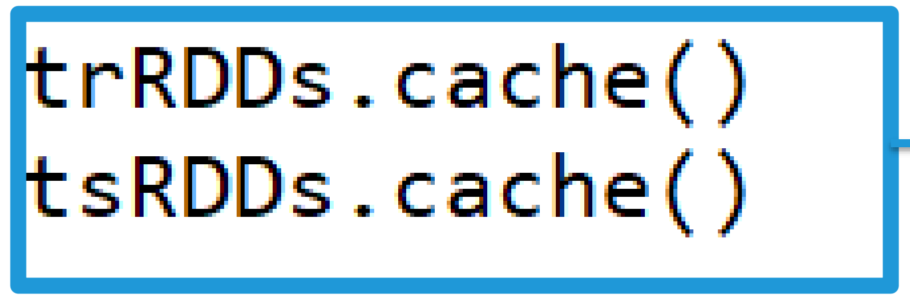


```
conf = SparkConf()
sc = SparkContext(conf=conf)
```

Configure Spark with SparkConf



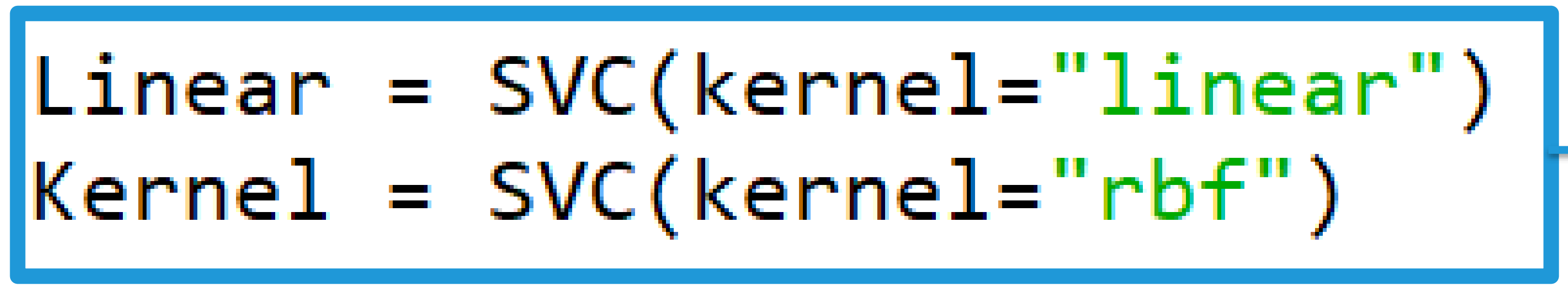
```
trRDDs = sc.parallelize(trX.tolist(), numPartition)
tsRDDs = sc.parallelize(tsX.tolist(), numPartition)
```



```
trRDDs.cache()
tsRDDs.cache()
```

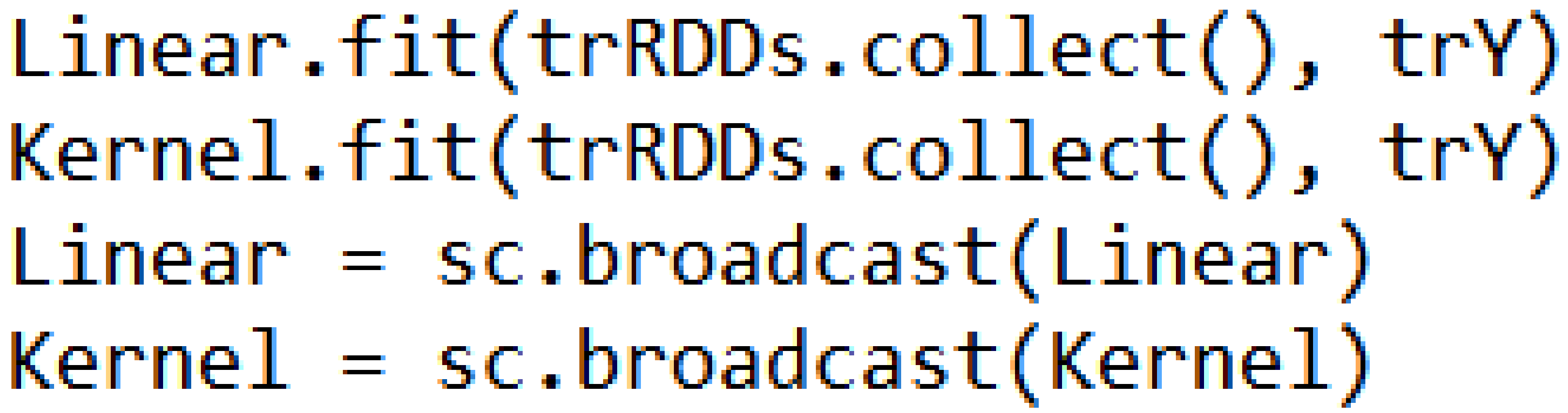
Save data into memory

- Train SVM model



```
Linear = SVC(kernel="linear")
Kernel = SVC(kernel="rbf")
```

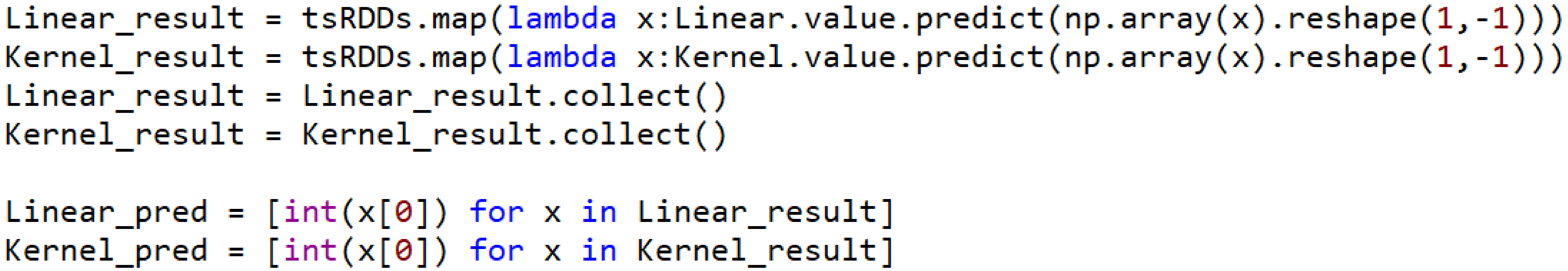
Configure “kernel” parameter for training linear model and kernel model



```
Linear.fit(trRDDs.collect(), trY)
Kernel.fit(trRDDs.collect(), trY)
Linear = sc.broadcast(Linear)
Kernel = sc.broadcast(Kernel)
```

Solution

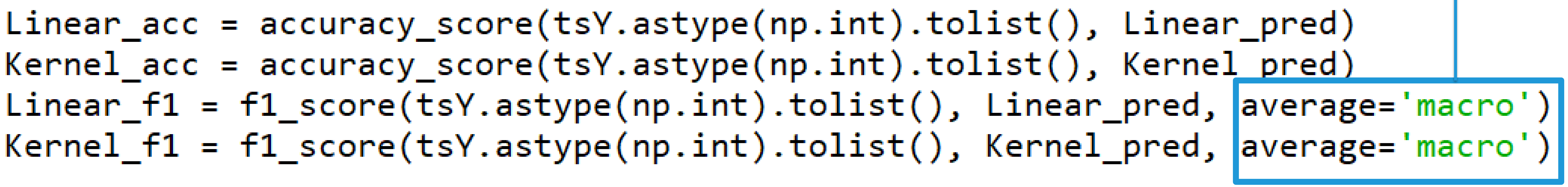
- ➢ **Predict class value of test dataset**



```
Linear_result = tsRDDs.map(lambda x:Linear.value.predict(np.array(x).reshape(1,-1)))
Kernel_result = tsRDDs.map(lambda x:Kernel.value.predict(np.array(x).reshape(1,-1)))
Linear_result = Linear_result.collect()
Kernel_result = Kernel_result.collect()

Linear_pred = [int(x[0]) for x in Linear_result]
Kernel_pred = [int(x[0]) for x in Kernel_result]
```

Configure “average” parameter to “macro” for averaging score of each labels.

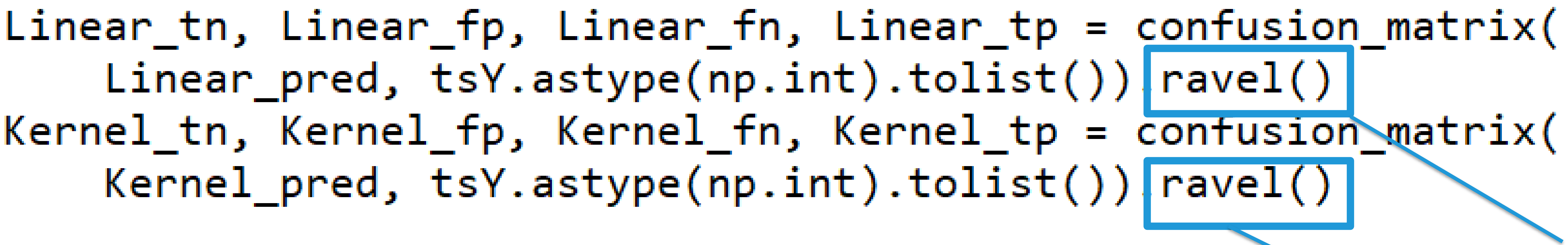
- ➢ **Calculate accuracy and F1 score**



```
Linear_acc = accuracy_score(tsY.astype(np.int).tolist(), Linear_pred)
Kernel_acc = accuracy_score(tsY.astype(np.int).tolist(), Kernel_pred)
Linear_f1 = f1_score(tsY.astype(np.int).tolist(), Linear_pred, average='macro')
Kernel_f1 = f1_score(tsY.astype(np.int).tolist(), Kernel_pred, average='macro')
```

Solution

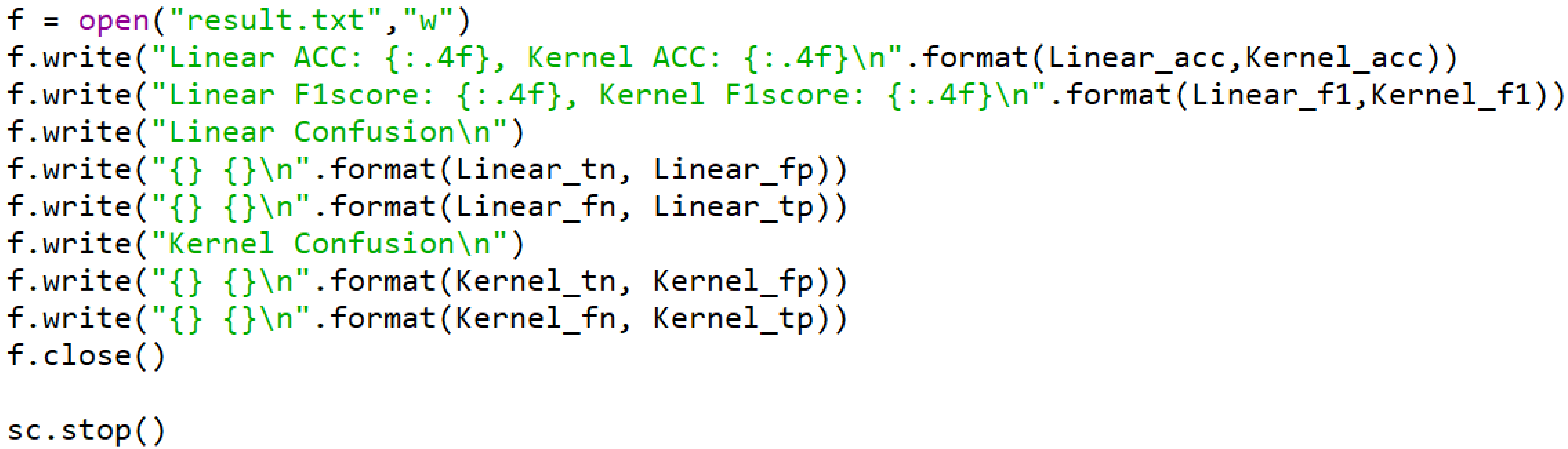
➢ Calculate confusion matrix



```
Linear_tn, Linear_fp, Linear_fn, Linear_tp = confusion_matrix(
    Linear_pred, tsY.astype(np.int).tolist()).ravel()
Kernel_tn, Kernel_fp, Kernel_fn, Kernel_tp = confusion_matrix(
    Kernel_pred, tsY.astype(np.int).tolist()).ravel()
```

Flatten multi-dimension arrays into one-dimension

➢ Save the result and quit Spark

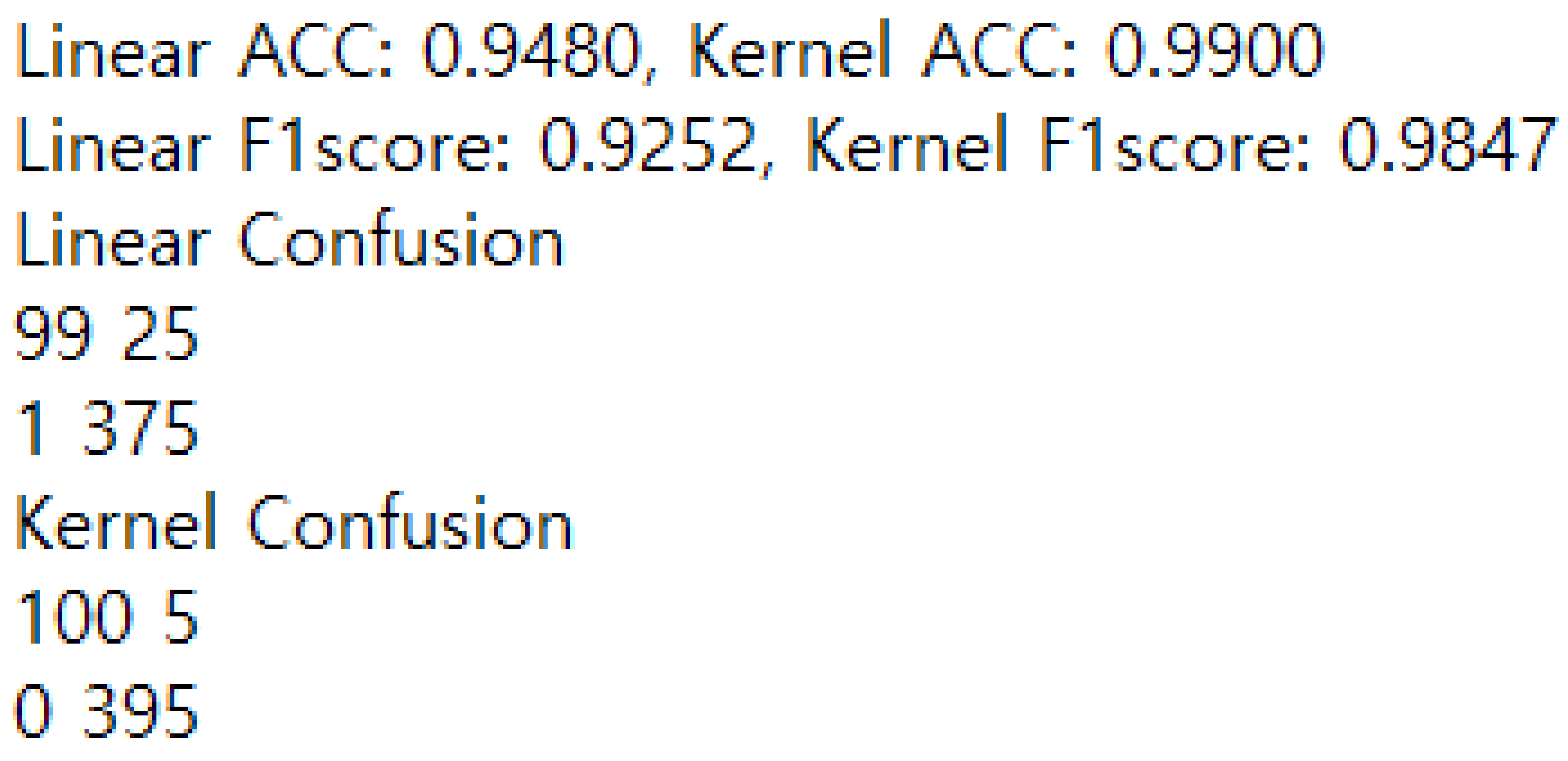


```
f = open("result.txt","w")
f.write("Linear ACC: {:.4f}, Kernel ACC: {:.4f}\n".format(Linear_acc,Kernel_acc))
f.write("Linear F1score: {:.4f}, Kernel F1score: {:.4f}\n".format(Linear_f1,Kernel_f1))
f.write("Linear Confusion\n")
f.write("{} {}\n".format(Linear_tn, Linear_fp))
f.write("{} {}\n".format(Linear_fn, Linear_tp))
f.write("Kernel Confusion\n")
f.write("{} {}\n".format(Kernel_tn, Kernel_fp))
f.write("{} {}\n".format(Kernel_fn, Kernel_tp))
f.close()

sc.stop()
```

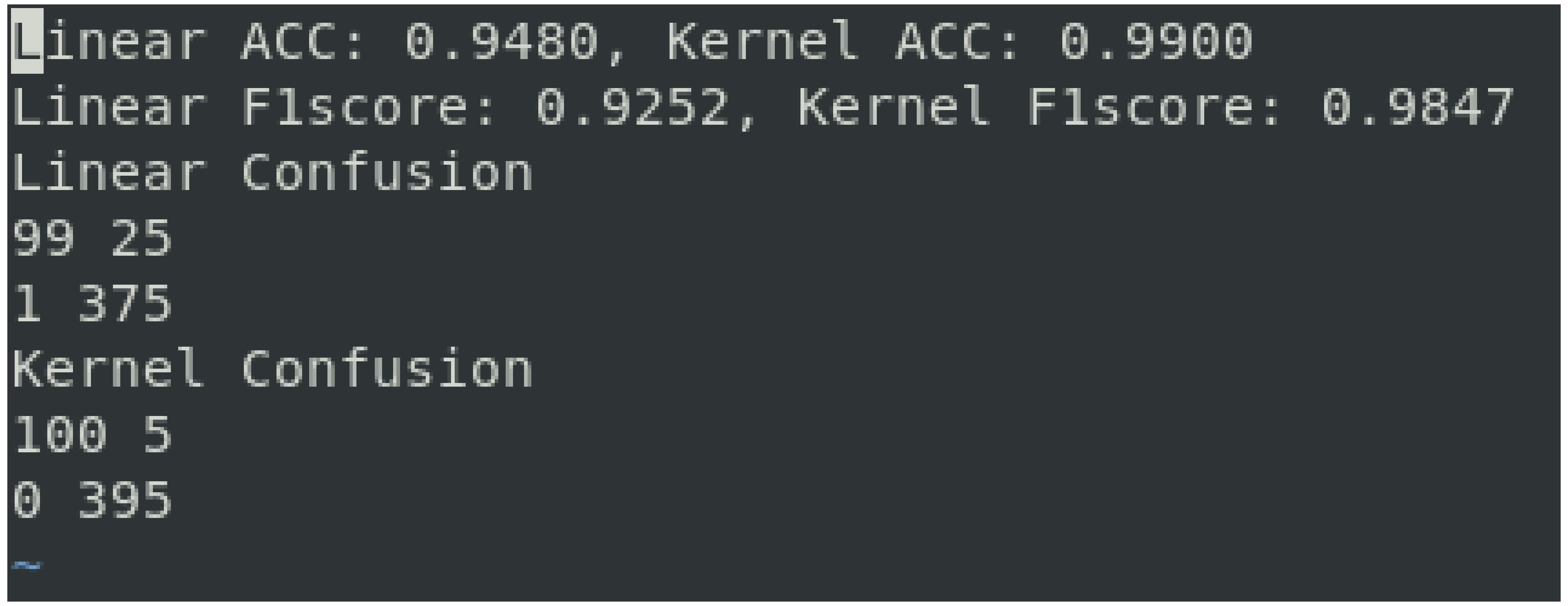
Solution

➢ Result

- Your result might be like the following



```
Linear ACC: 0.9480, Kernel ACC: 0.9900
Linear F1score: 0.9252, Kernel F1score: 0.9847
Linear Confusion
99 25
1 375
Kernel Confusion
100 5
0 395
```



```
Linear ACC: 0.9480, Kernel ACC: 0.9900
Linear F1score: 0.9252, Kernel F1score: 0.9847
Linear Confusion
99 25
1 375
Kernel Confusion
100 5
0 395
~
```

Windows

Linux