DS-630 Week 4 Assignment

Please complete the below problems.

1. Suppose you've trained an SVM classifier with an RBF kernel, but it seems to underfit the training set. Should you increase or decrease γ (gamma)? What about C? Explain your reasoning.
2. Train an SVM classifier on the MNIST dataset. Since SVM classifiers are binary classifiers, you will need to use one-versus-the-rest to classify all 10 digits. You may want to tune the hyperparameters using small validation sets to speed up the process. What accuracy can you reach?

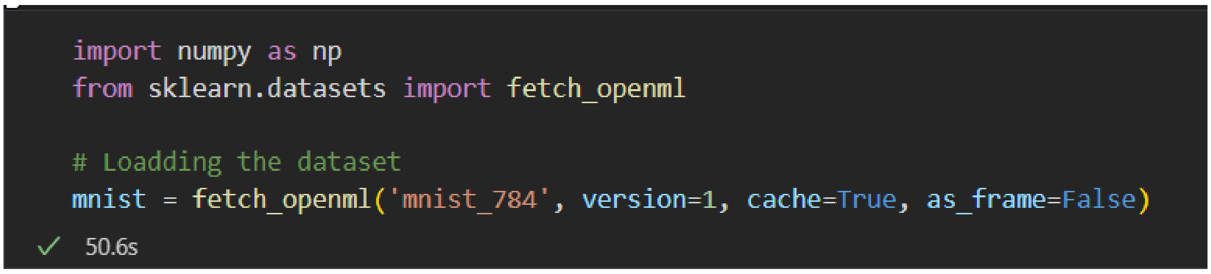
For each weekly assignment, please write a summary, analysis, and key findings and attach codes to your submission as necessary. Feedback to your assignments will be provided in the grade book.

Due date: Sunday of Week 4, 11:59 p.m. EST.

1. The underfit data set might be because of too much regularization. To reduce underfit need to increase Gamma or C or Both.

2.

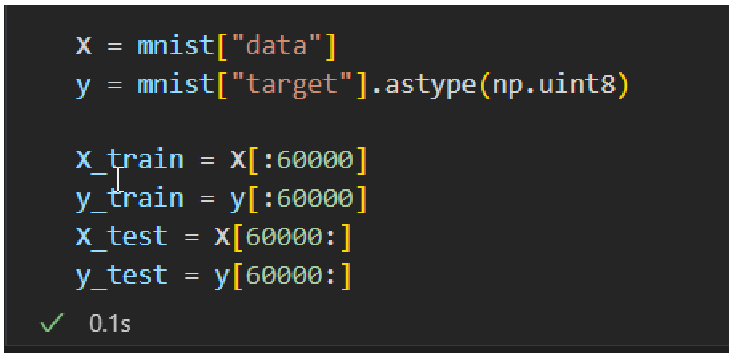
Loading data set



```
import numpy as np
from sklearn.datasets import fetch_openml

# Loadding the dataset
mnist = fetch_openml('mnist_784', version=1, cache=True, as_frame=False)
```
✓ 50.6s

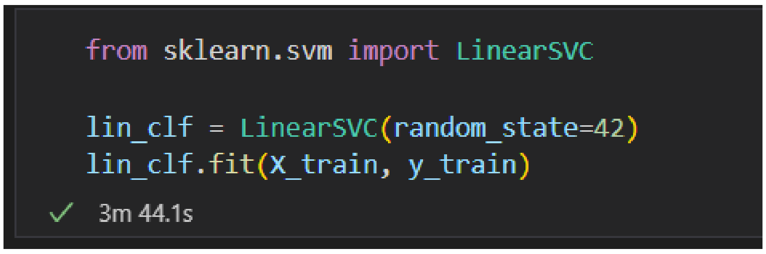
Splitting into training and testing datasets set. people usually just take the first 60,000 instances for the training set and the last 10,000 instances for the test set.



```
X = mnist["data"]
y = mnist["target"].astype(np.uint8)

X_train = X[:60000]
y_train = y[:60000]
X_test = X[60000:]
y_test = y[60000:]
```
✓ 0.1s

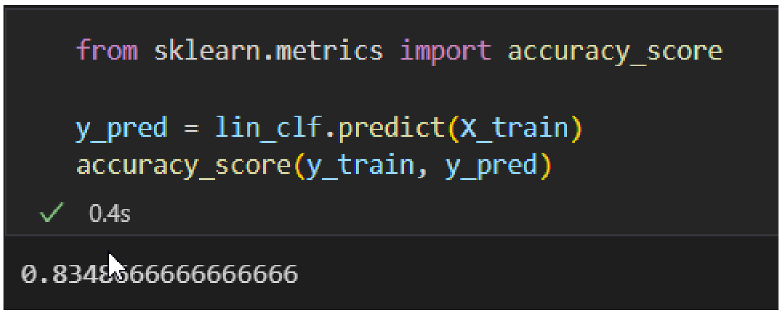
It's common practice to shuffle the training instances first because many training algorithms are sensitive to their order. However, since the dataset has already been scrambled, we do not need to. Start off easy by using a linear SVM classifier. There is nothing additional we need to do because it will automatically utilize the One-vs-All (also known as One-vs-the-Rest, OvR) strategy. Easy!



```
from sklearn.svm import LinearSVC

lin_clf = LinearSVC(random_state=42)
lin_clf.fit(X_train, y_train)
```
✓ 3m 44.1s

Making predictions and finding accuracy

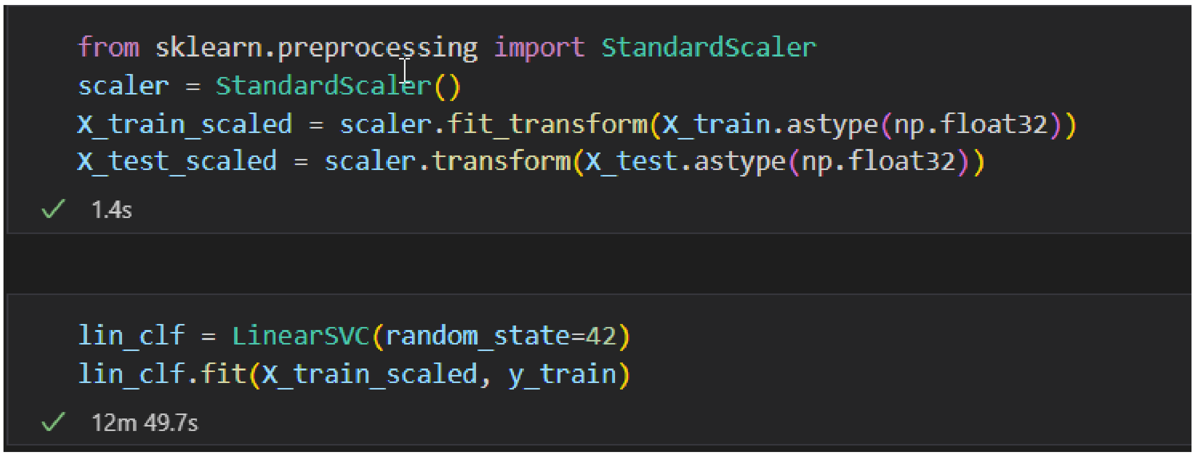


```
from sklearn.metrics import accuracy_score

y_pred = lin_clf.predict(X_train)
accuracy_score(y_train, y_pred)
```
✓ 0.4s

```
0.8348666666666666
```

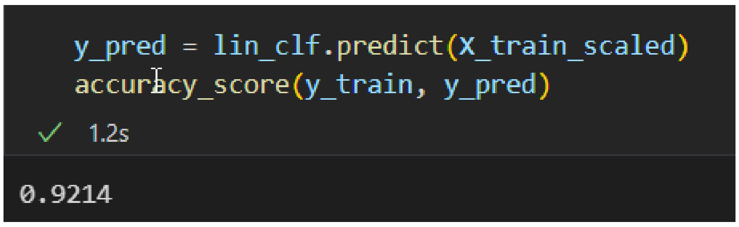
Linear model is certainly too simple for most. for MNIST 83% accuracy is too low.



```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float32))
X_test_scaled = scaler.transform(X_test.astype(np.float32))
```
✓ 1.4s

```
lin_clf = LinearSVC(random_state=42)
lin_clf.fit(X_train_scaled, y_train)
```
✓ 12m 49.7s

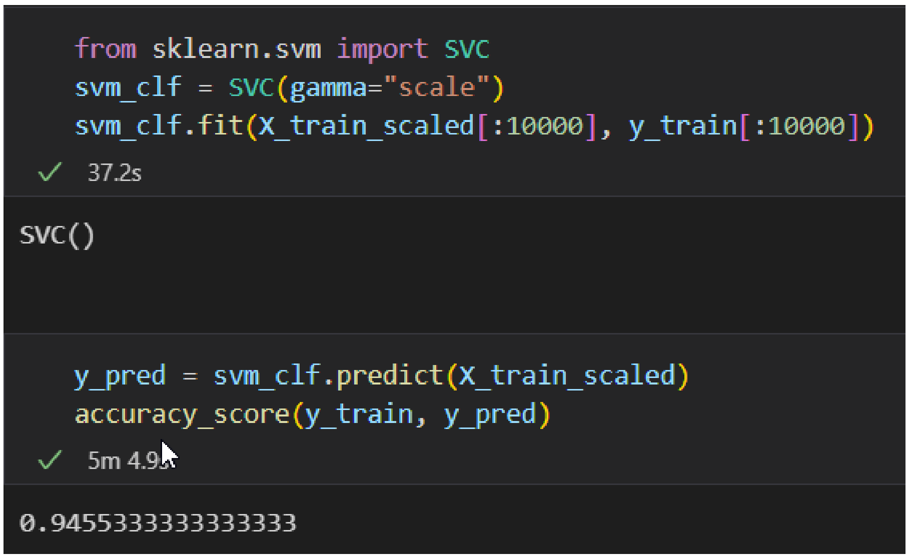
this accuracy is better than linear but still not good enough for MNIST.



```
y_pred = lin_clf.predict(X_train_scaled)
accuracy_score(y_train, y_pred)
```
✓ 1.2s

```
0.9214
```

That's encouraging because performance has improved despite the model having been trained on six times less. Let's perform a randomized search using cross-validation to adjust the hyperparameters. To expedite the procedure, we'll perform this on a small dataset:

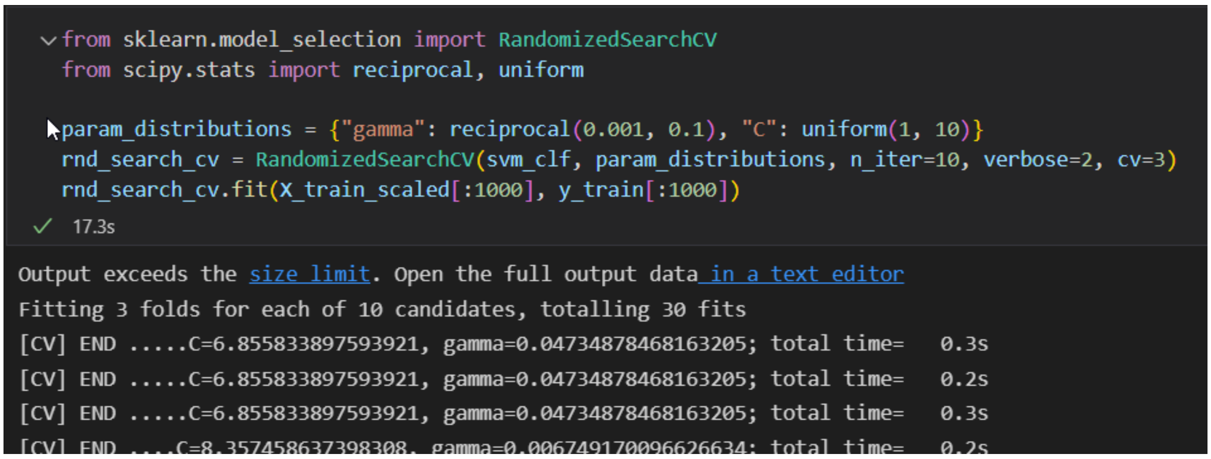


```
from sklearn.svm import SVC
svm_clf = SVC(gamma="scale")
svm_clf.fit(X_train_scaled[:10000], y_train[:10000])
```
✓ 37.2s

```
SVC()
```

```
y_pred = svm_clf.predict(X_train_scaled)
accuracy_score(y_train, y_pred)
```
✓ 5m 4.9s

```
0.9455333333333333
```

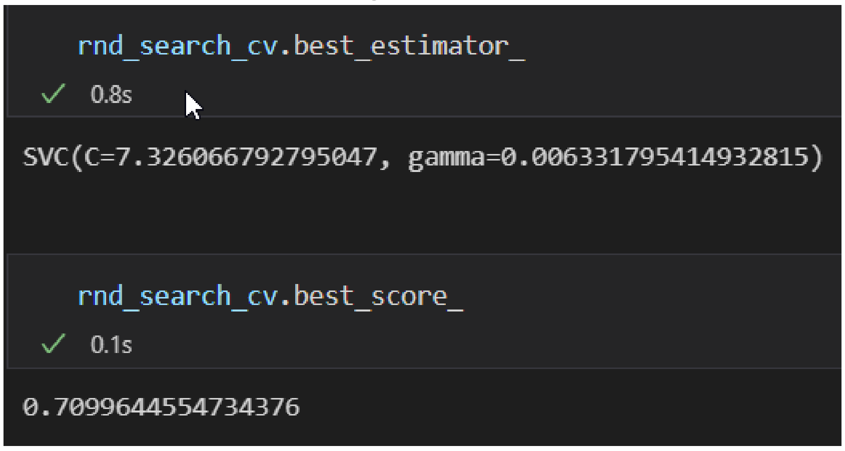


```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import reciprocal, uniform

param_distributions = {"gamma": reciprocal(0.001, 0.1), "C": uniform(1, 10)}
rnd_search_cv = RandomizedSearchCV(svm_clf, param_distributions, n_iter=10, verbose=2, cv=3)
rnd_search_cv.fit(X_train_scaled[:1000], y_train[:1000])
```
✓ 17.3s

```
Output exceeds the size limit. Open the full output data in a text editor
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[CV] END .....C=6.855833897593921, gamma=0.04734878468163205; total time=   0.3s
[CV] END .....C=6.855833897593921, gamma=0.04734878468163205; total time=   0.2s
[CV] END .....C=6.855833897593921, gamma=0.04734878468163205; total time=   0.3s
[CV] END .....C=8.357458637398308, gamma=0.006749170096626634; total time=   0.2s
```

Although it seems low, keep in mind that we only trained the model on 1,000 occasions.
Retrained on the entire training set, let's use the best estimator:



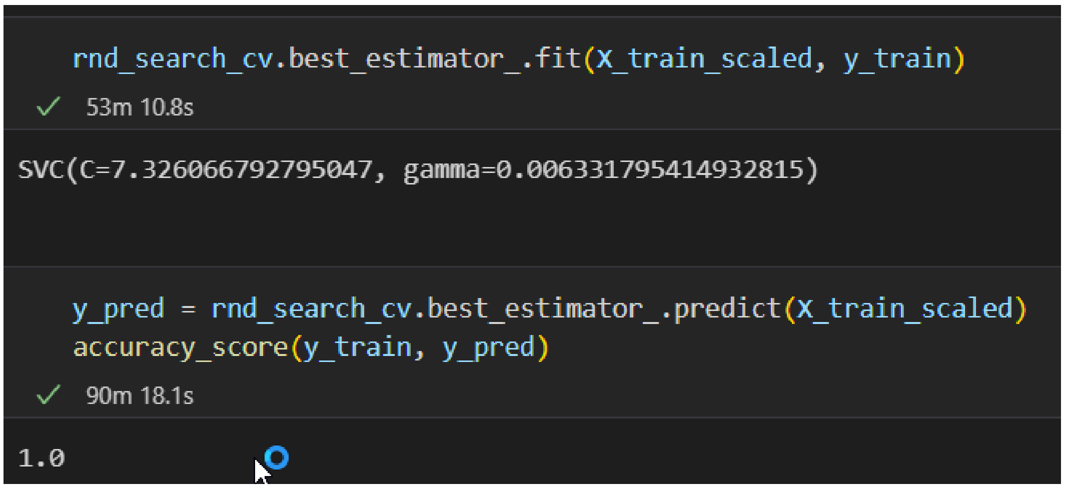
```
rnd_search_cv.best_estimator_
```
✓ 0.8s

```
SVC(C=7.326066792795047, gamma=0.006331795414932815)
```

```
rnd_search_cv.best_score_
```
✓ 0.1s

```
0.7099644554734376
```

This is better than another model. but looks like overfitting model.



```
rnd_search_cv.best_estimator_.fit(X_train_scaled, y_train)
```

✓ 53m 10.8s

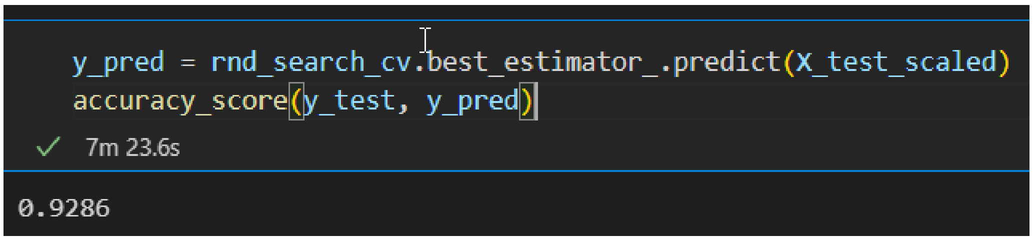
```
SVC(C=7.326066792795047, gamma=0.006331795414932815)
```

```
y_pred = rnd_search_cv.best_estimator_.predict(X_train_scaled)
accuracy_score(y_train, y_pred)
```

✓ 90m 18.1s

```
1.0
```

Testing on the test set using random search. It’s not bad but it looks like the model is slightly overfitting.



```
y_pred = rnd_search_cv.best_estimator_.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```

✓ 7m 23.6s

```
0.9286
```

Based on the code I can say that random search is a good model but might be its slightly over fitting.