2η εργασία στα Νευρωνικά δίκτυα

Ιωακείμ Ευαγγελινός (4187)

January 24, 2025

1 Ι. Εισαγωγή

Κατά την εκπόνηση της παρούσας εργασία, εκπαιδεύονται και συγκρίνονται 39 μοντέλα SVM, το καθένα με διαφορετικές υπέρ-παραμέτρους, στο σύνολο δεδομένων των κλάσεων cat και ship του CIFAR10 [1]. Συγκεκριμένα, συγκρίνονται τα ποσοστά επιτυχίας στην Ακρίβεια (Accuracy) στα σύνολα δεδομένων εκπαίδευσης και δοκιμής και οι χρόνοι εκπαίδευσης. Έπειτα, συγκρίνονται με τους κατηγοριοποιητές KNN για K=3 και K=5, NC και με MLP με 1 hidden layer με Hinge Loss.

2 Υπόβαθρο

Το CIFAR10 είναι ένα σύνολο δεδομένων 60000 RGB εικόνων με διαστάσεις $32x32$. Το σύνολο δεδομένων χωρίζεται σε σύνολο δεδομένων εκπαίδευση (50000 εικόνες) και δοκιμής (10000 εικόνες). Χρησιμοποιείται εκτενώς για εκπαιδευτικούς σκοπούς με στόχο την σωστή κατηγοροποίηση μιας εικόνας στην σωστή κατηγορία.
Το PyTorch είναι μια πολύ χρήσιμη βιβλιοθήκη της Python για την δημιουργία νευρωνικών δικτύων.
Το Scikit-learn είναι μια πολύ χρήσιμη βιβλιοθήκη της Python για την δημιουργία ΣΜ που χρησιμοποιεί την λιβσμ για τα Σ

3 Προεπεξεργασία δεδομένων

Κατά την φόρτωση των δεδομένων, εφαρμόζω τα transformations που έκανα στην 1η εργασία και διαγράφω όλα τα δείγματα που δεν είναι γάτα ή πλοίο. Στην συνέχεια, μετατρέπω τα δεδομένα σε μορφή που είναι αποδεκτή από τα SVC του Scikit-learn.

4 Αλγόριθμος

Δημιουργώ δυναμικά μοντέλα SVM με βάση τις παραμέτρους που θέλω να έχουν και τα συγκρίνω. Ο πίνακας 1 περιέχει τα αποτελέσματά τους. Ο παρακάτω κώδικας περιγράφει την διαδικασία:

```
epoches=8000
svms={}
Cs=[1,0.1,0.001]
dfs=["ovr"]
kernels=["linear", "poly", "rbf", "sigmoid"]
gammas=["auto","scale",0.001]
class_weights=[None]
degrees=[3,5]
tols=[0.001]
for tol in tols:
  for c in Cs:
    for df in dfs:
      for kernel in kernels:
        for class_weight in class_weights:
          if (kernel=="poly"):
```

```
            for degree in degrees:
              for gamma in gammas:
                svm_name="kernel_{0}/c_{1}/cw_{2}/dfs_{3}/tol_{4}/gamma_{5}/degree_{6}"
                .format(kernel,c,class_weight,df,tol,gamma,degree)
                svms[svm_name]=SVC(kernel=kernel,
                ↪  C=c,decision_function_shape=df,gamma=gamma,degree=degree,
                tol=tol,class_weight=class_weight, max_iter=epoches)
                Path("/home/i/ioakeime/Models/svms/"+svm_name).mkdir(parents=True,
                ↪  exist_ok=True)
          elif (kernel=="sigmoid" or kernel=="rbf"):
            for gamma in gammas:
              svm_name="kernel_{0}/c_{1}/cw_{2}/dfs_{3}/tol_{4}/gamma_{5}"
              .format(kernel,c,class_weight,df,tol,gamma)
              svms[svm_name]=SVC(kernel=kernel,
              ↪  C=c,decision_function_shape=df,gamma=gamma,
              tol=tol,class_weight=class_weight, max_iter=epoches)
              Path("/home/i/ioakeime/Models/svms/"+svm_name)
              .mkdir(parents=True, exist_ok=True)
          else:
            svm_name="kernel_{0}/c_{1}/cw_{2}/dfs_{3}/tol_{4}/".
            format(kernel,c,class_weight,df,tol)
            svms[svm_name]=SVC(kernel=kernel,
            ↪  C=c,decision_function_shape=df,tol=tol,class_weight=class_weight,
            ↪  max_iter=epoches)
            Path("/home/i/ioakeime/Models/svms/"+svm_name).mkdir(parents=True,
            ↪  exist_ok=True)

```

Το MLP και οι κατηγοροποιητές έγιναν με τον ίδιο τρόπο όπως στην 1η εργασία.

Μοντέλο MLP:

$Linear \rightarrow ReLu \rightarrow\rightarrow Linear \rightarrow Softmax$

5 Ανάλυση μοντέλων SVM

Ακολουθούν οι ευρέσεις μου για τον κάθε kernel:

linear

Για τα μοντέλα με kernel linear, από τα δεδομένα του πίνακα, καταλαβαίνω ότι η σύγκλιση είναι αργή και εξαρτάται πολύ από την παράμετρος C (αφού δεν σύγκλινε κανένα μοντέλο και το μοντέλο πιο κοντά σε σύγκλιση ήταν αυτό με το μικρότερο C). Ο χρόνος εκπαίδευσης αυξάνεται όσω μειώνεται το C. Η απόδοση φαίνεται να αυξάνεται όσο μειώνεται το C. Ο μέσος χρόνος εκπαίδευσης ήταν περίπου 113 δευτερόλεπτα.

sigmoid

Φαίνεται να επηρεάζεται αρκετά από το C. Το gamma φαίνεται να το επηρεάζει αρκετά λιγότερο από το το C. Οι τιμές auto και gamma φαίνονται να έχουν παρόμοιες αποδόσεις καθώς δεν κυριαρχεί κάποιο από τα 2. Το gamma 0.0001 δεν αποδίδει τόσο καλά σε σύγκριση με τις άλλες 2 τιμές. Η απόδοση φαίνεται να αυξάνεται όσο μειώνεται το C. Ο χρόνος εκπαίδευσης φαίνεται να αυξάνεται όσο μικρύνεται το C. Ο μέσος χρόνος εκπαίδευσης ήταν περίπου 120 δευτερόλεπτα. Τα μοντέλα συγκλίνουν με τις λιγότερες επαναλήψεις σε σύγκριση με τα άλλα kernel. Όλα τα μοντέλα με sigmoid συγκλίνουν.

rbf

Φαίνεται να επηρεάζεται αρκετά από το C. Η απόδοση φαίνεται να αυξάνεται όσο αυξάνεται το C. Το gamma φαίνεται να το επηρεάζει αρκετά λιγότερο από το το C. Οι τιμές auto και gamma φαίνονται να έχουν παρόμοιες αποδόσεις καθώς δεν κυριαρχεί κάποιο από τα 2. Το gamma 0.001 δεν αποδίδει τόσο καλά σε σύγκριση με τις άλλες 2 τιμές. Ο χρόνος εκπαίδευσης φαίνεται να αυξάνεται όσο μικρύνεται το C. Οι επαναλήψεις που χρειάζονται για σύγκλιση μειώνονται όσο μειώνεται το C. Ο μέσος χρόνος εκπαίδευσης ήταν περίπου 126 δευτερόλεπτα.

poly

Οι τιμές auto και gamma φαίνονται να έχουν παρόμοιες αποδόσεις καθώς δεν κυριαρχεί κάποιο από τα 2. Το gamma 0.001 δεν αποδίδει τόσο καλά σε σύγκριση με τις άλλες 2 τιμές αλλά η διαφορά φαίνεται

μικρότερη από άλλα kernel. Τα μεγάλα και μικρά C φαίνεται σε γενικές γραμμές να έχουν χειρότερη απόδοση από το 0.1, αν και τα μεγάλα C λειτουργούν καλύτερα από τα μικρά. Όσο μικραίνει το C, τόσο πιο γρήγορα συγκλίνει. Το degree 3 αποδίδει καλύτερα αποτελέσματα από 5. Δεν μπόρεσα να προσέξω κάποια σχέση μεταξύ των υπερπαραμέτρων με τον χρόνο εκπαίδευσης. Ο μέσος χρόνος εκπαίδευσης ήταν περίπου 125 δευτερόλεπτα.
Το καλύτερο μοντέλο είναι το 8ο με 0.868 accuracy (αν και τείνει στο να γίνει overfitted)

Table 1: SVM Comparison

Kernel	**C**	**Tol**	**Gamma**	**Degree**	**Train_acc**	**Test_acc**	**Time_secs**	**Acc_cat**	**Acc_ship**	**Iterations**	**Converged**
linear	1	0.001	0	0	0.5395	0.504	62.9274	0.504	0.504	8000	False
poly	1	0.001	auto	3	0.9577	0.683	119.6071	0.683	0.683	8000	False
poly	1	0.001	scale	3	0.8508	0.725	129.757	0.725	0.725	8000	False
poly	1	0.001	0.001	3	0.8842	0.592	92.948	0.592	0.592	8000	False
poly	1	0.001	auto	5	0.8483	0.555	108.538	0.555	0.555	8000	False
poly	1	0.001	scale	5	0.8759	0.707	145.6832	0.707	0.707	8000	False
poly	1	0.001	0.001	5	0.8128	0.5345	79.3355	0.5345	0.5345	8000	False
rbf	**1**	**0.001**	**auto**	**0**	**0.961**	**0.868**	**86.815**	**0.868**	**0.868**	**8000**	**False**
rbf	1	0.001	scale	0	0.9319	0.863	81.016	0.863	0.863	6749	True
rbf	1	0.001	0.001	0	0.9943	0.8245	161.5344	0.8245	0.8245	8000	False
sigmoid	1	0.001	auto	0	0.561	0.579	107.7995	0.579	0.579	5329	True
sigmoid	1	0.001	scale	0	0.5723	0.586	106.8842	0.586	0.586	6932	True
sigmoid	1	0.001	0.001	0	0.5134	0.5325	86.9114	0.5325	0.5325	3513	True
linear	0.1	0.001	0	0	0.5395	0.504	61.5824	0.504	0.504	8000	False
poly	0.1	0.001	auto	3	0.785	0.718	128.8261	0.718	0.718	8000	False
poly	0.1	0.001	scale	3	0.7183	0.6985	133.3068	0.6985	0.6985	6866	True
poly	0.1	0.001	0.001	3	0.9393	0.621	104.6276	0.621	0.621	8000	False
poly	0.1	0.001	auto	5	0.8892	0.7125	143.1497	0.7125	0.7125	8000	False
poly	0.1	0.001	scale	5	0.6727	0.6235	146.0265	0.6235	0.6235	8000	False
poly	0.1	0.001	0.001	5	0.8184	0.536	81.206	0.536	0.536	8000	False
rbf	0.1	0.001	auto	0	0.8252	0.8065	105.0626	0.8065	0.8065	4094	True
rbf	0.1	0.001	scale	0	0.8205	0.801	102.3941	0.801	0.801	3955	True
rbf	0.1	0.001	0.001	0	0.6846	0.67	141.5823	0.67	0.67	5175	True
sigmoid	0.1	0.001	auto	0	0.5621	0.58	117.8569	0.58	0.58	5631	True
sigmoid	0.1	0.001	scale	0	0.5801	0.5955	130.9691	0.5955	0.5955	7463	True
sigmoid	0.1	0.001	0.001	0	0.51	0.529	113.7114	0.529	0.529	4785	True
linear	0.001	0.001	0	0	0.7399	0.7055	100.1849	0.7055	0.7055	8000	False
poly	0.001	0.001	auto	3	0.6031	0.6005	143.9571	0.6005	0.6005	4821	True
poly	0.001	0.001	scale	3	0.56	0.561	148.15	0.561	0.561	4972	True
poly	0.001	0.001	0.001	3	0.7258	0.7015	133.1379	0.7015	0.7015	7179	True
poly	0.001	0.001	auto	5	0.5807	0.5765	145.9212	0.5765	0.5765	5854	True
poly	0.001	0.001	scale	5	0.5322	0.53	146.3314	0.53	0.53	5000	True
poly	0.001	0.001	0.001	5	0.896	0.5915	132.7398	0.5915	0.5915	8000	False
rbf	0.001	0.001	auto	0	0.6826	0.6845	156.1029	0.6845	0.6845	5000	True
rbf	0.001	0.001	scale	0	0.6986	0.7105	152.4645	0.7105	0.7105	5000	True
rbf	0.001	0.001	0.001	0	0.5339	0.533	152.679	0.533	0.533	5000	True
sigmoid	0.001	0.001	auto	0	0.6558	0.6735	156.4322	0.6735	0.6735	5060	True
sigmoid	0.001	0.001	scale	0	0.6528	0.6705	157.5546	0.6705	0.6705	5038	True
sigmoid	0.001	0.001	0.001	0	0.6347	0.6555	151.4138	0.6555	0.6555	5045	True

6 Σύγκριση

Ακολουθεί ο πίνακας σύγκρισης των μοντέλων/κατηγοριοποιητών που χρειάζονται σύγκριση:

Από τον πίνακα, βλέπουμε ότι όταν έχουμε μειώσει τις κλάσεις σε 2, έχουν βελτιωθεί σημαντικά η ακρίβεια όλων των κατηγοριοποιητών KNN και NC.
Από τα δεδομένα, ταξινομώ τα μοντέλα από καλύτερο προς χειρότερο ως εξής:
$SVM \rightarrow 3-NN \rightarrow 1-NN \rightarrow MLP \rightarrow NC$

Table 2: Comparison of the models/classifiers of all categories

Classifier/Model	Train_time	Test_time	Accuracy
KNN(K=1)	0.1533	1.0619	0.747
KNN(K=3)	0.0562	0.7066	0.761
NC	0.10159	0.0299	0.6615
MLP(1 hidden)	80.0814	0.4209	0.7425
Best SVM	86.815	40.6294	0.868

Βλέπουμε ότι λόγο της μικρής περιπλοκότητας του MLP, δεν τα πάει τόσο καλά σε σύγκριση με τα άλλα. Το SVM κυριαρχεί τα μοντέλα ένα πολύ ικανοποιητικό accuracy αλλά αρκετά αργό training και testing. Οι κατηγοριοποιητές μπόρεσαν να σταθούν δυνατά στο μειωμένο σύνολο δεδομένων, σε σύγκριση με την 1η εργασία.
Συμπερασματικά, θα χρησιμοποιούσα το SVM από όλα τα μοντέλα, και αν υπάρχουν σοβαρές απαιτήσεις χρόνου, θα επέλεγα το 3-NN.

References

[1] *Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.* https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf