Documentație IA

Agha Mara, Grupa 243

6 Aprilie 2021

I k-NN

```
class KnnClassifier:
    def __init__(self, train_images, train_labels):
        self.train_images = train_images
        self.train_labels = train_labels

    def classify_images(self, test_image, num_neighbors, metric):
        # pt fiecare imagine test i, avem cate o distanta

        # distanta Euclidiana
        if metric == 'l2':
            distances = np.sum((train_images - test_image) ** 2, axis=-1)
        # distanta Manhattan
        elif metric == 'l1':
            distances = np.sum(np.abs(train_images - test_image), axis=-1)

        # sortare crescatoare dupa distante, sunt reordonati si returnati indecsii
        # functia argsort returneaza indecsii care ordoneaza lista
        sorted_indexes = np.argsort(distances)
        # sunt luate primele 3 etichete din indecsii sortati
        top_neighbors = self.train_labels[sorted_indexes[:num_neighbors]]
        # functia bincount calculeaza nr de aparitii al fiecarei valori din lista
        class_counts = np.bincount(top_neighbors)

        return np.argmax(class_counts)


clf = KnnClassifier(train_images, train_labels)

def test(k, metric):
    predictions = []
    for validation_image in validation_images:
        pred_label = clf.classify_images(validation_image, k, metric)
        predictions.append(pred_label)

    pred_labels = np.array(predictions)
    correct_count = np.sum(pred_labels == validation_labels)
    total_count = len(validation_labels)

    accuracy = correct_count / total_count

```

```
    print("Accuracy for " + str(k) + " neighbours and " + metric + " metric: " + str
    (accuracy*100) + "%")


for k in range(3, 60, 2):
    test(k, "l1")
    test(k, "l2")
```

Acuratețile obținute		
K	Metric	
	L1	L2
3	24.02%	22.34%
5	24.94%	22.88%
7	24.0000000000002%	24.12%
9	24.58%	24.92%
11	24.38%	25.380000000000003%
13	24.0%	25.1%
15	24.099999999999998%	25.16%
17	24.240000000000002%	25.31999999999997%
19	24.08%	25.18000000000003%
21	23.96%	25.08000000000002%
23	23.84%	24.84%
25	23.82%	24.64%
27	23.68%	24.92%

Calculează distanțele dintre o probă și restul datelor, selectând numărul specificat de exemple (k) cele mai apropiate de probă, apoi îi atribuie clasa cea mai frecventă.

II SVM

```
# Importanta normalizarii: datele sa fie aduse la aceeasi scara
# asa incat sa fie compatibile
def normalize_data(train_data, test_data, norm_type):
    if norm_type is None:
        return train_data, test_data

    # Transforma vectorii de caracteristici astfel incat fiecare
    # sa aiba medie 0 si deviatie standard 1
    if norm_type == 'standard':
        scaler = preprocessing.StandardScaler()
        scaler.fit(train_data)
        return scaler.transform(train_data), scaler.transform(test_data)

    # Transforma fiecare caracteristica individual intre 0 si
    # 1, ceea ce e o metoda buna pentru a pastra valorile de
    # 0 intr-un set de date imprastiat (si acestea pot fi
    # eliminate ulterior)
    if norm_type == 'min_max':
        return (preprocessing.minmax_scale(train_data, axis=-1),
                preprocessing.minmax_scale(test_data, axis=-1))

    # Varianta mai robusta decat L2 deoarece aici se iau doar
    # valorile absolute, deci le trateaza liniar
    if norm_type == 'l1':
        return (preprocessing.normalize(train_data, norm='l1'),
                preprocessing.normalize(test_data, norm='l1'))

    # Varianta mai stabila decat L1 (rezistenta mai mare la
    # ajustari orizontale)
    if norm_type == 'l2':
        return (preprocessing.normalize(train_data, norm='l2'),
                preprocessing.normalize(test_data, norm='l2'))


def test(norm, kernel, c):
    X_train, X_test = normalize_data(train_images, test_images, norm)
    X_train1, X_validation = normalize_data(train_images, validation_images, norm)
    if kernel == 'linear':
        clf = SVC(C = c, kernel = 'linear')
    else:
        clf = SVC(kernel='rbf', gamma=c/10)
    hist = clf.fit(X_train, train_labels)
    preds = clf.predict(X_validation)
    accuracy = accuracy_score(validation_labels, preds)

    print("Accuracy " + norm + " norm with " + kernel + " kernel and " + str(c) +"
    parameter:", accuracy)


for i in range(3,16,2):
    test('l1', 'rbf', i)
    test('l2', 'rbf', i)
    test('min_max', 'rbf', i)
    test('standard', 'rbf', i)

    test('l1', 'linear', i)
```

```
    test('l2', 'linear', i)
    test('min_max', 'linear', i)
    test('standard', 'linear', i)
```

Optimizarea acestui clasificator constă în:

- Maximizarea marginii dintre vectorii suport (extremitățile unui cluster dintr-o anumită clasă de date)
- Maximizarea numărului de date corect clasificate

A doua strategie de optimizare prezintă un mare risc de overfit, așa că se va prefera misclasificarea anumitor date pentru a crea un model mai general care să ofere o acuratețe bună inclusiv pe datele de validare. Această opțiune se implementează prin normalizarea datelor, ajustarea hiperparametrilor și alegerea tipului de kernel.

Acuratețile obținute						
Kernel	Parametru	Valoare	Norm			
			L1	L2	min_max	standard
linear	C	3	0.2896%	0.625%	0.5856%	0.5574%
		5	0.377%	0.6282%	0.5764%	0.5512%
		7	0.398%	0.6292%	0.5718%	0.5484%
		9	0.4128%	0.6314%	0.5472%	0.5694%
rbf	gamma	0.3	0.1108%	0.63065	0.1272%	0.1116%
		0.5	0.1338%	0.6524%	0.1186%	0.111%
		0.7	0.179%	0.6708%	0.1148%	0.1108%
		0.9	0.2186%	0.6812%	0.1134%	0.1108%

Importanța tipului de kernel: proiectează datele non-liniar separabile dintr-un spațiu m-dimensional într-un spațiu n-dimensional (m < n) unde datele devin liniar separabile deoarece punctele ce aparțin claselor diferite vor fi alocate unor dimensiuni diferite. Astfel, SVM-ul calculează hiperplanul într-un spațiu caracteristic multidimensional fără a transforma efectiv datele (ceea ce ar fi o operație costisitoare)

Hiperparametrii:

- Pentru kernel liniar: C
  - penalizarea pentru fiecare dată misclasificată

  - un C mic reprezintă un hiperplan cu o margine mai mare și un număr mai mare de date misclasificate
  - un C mare reprezintă încercarea SVM-ului de a minimiza numărul de date misclasificate prin alegerea unui hiperplan cu margini cât mai mici
  - acest factor de penalizare nu este acelasi pentru toate datele misclasificate, el este direct proporțional cu distanța până la hiperplan
  - De obicei C $\in$ (0.1, 100)

- Pentru kernel Gaussian (RBF): gamma (C a rămas default 1)
  - specific kernelul-ui RBF
  - controlează distanța de influență a unui unic punct de antrenare
  - un gama mic indică o rază mare a similitudinii ceea ce duce la gruparea mai multor clase de date (generalizarea graniței delimitatoare a claselor)
  - un gama mare implică o apropiere mai strânsă a punctelor pentru a putea fi considerate din aceeași clasă și asta implică un risc de overfit (orice distorsionare adusă unei date necunoscute modelului poate duce la misclasificarea acesteia)
  - de obicei gamma $\in$ (0.0001, 10)

III CNN

```
def build_model(test_images):
    # Model secvential care permite stratificarea layerelor
    model = Sequential()

    # Strat convolutional, de input
    # Dimensiunea kernel-ului este impara asa incat sa se centreze
    # peste regiunea imaginii asupra careia aplica produsul scalar
    # Am ales sa construiesc un model simplu, cu cat mai putini
    # parametrii antrenabili, asa incat sa asigur generalitatea
    # modelului. Kernelul de dimensiune 5 si cele 32 de filtre
    # vor invata la inceput caracteristici de baza din setul de date
    # precum linii verticale si orizontale
    # Functia de regularizare a kernelului se aplica matricei de ponderi
    # Hiperparametrul dat distantei euclidiene a fost ales asa
    # incat sa fie cat mai mic posibil, existand o preferinta pentru
    # ponderile mici la penalizare
    # Functia de activare: ReLU
    # Avantajele utilizarii functiei ReLU: imprastierea si un grad redus
    # de disparitie a gradientului
    model.add(Conv2D(32, (5, 5), activation='relu', input_shape=(32, 32, 1),
                     kernel_regularizer=tf.keras.regularizers.l2(0.001)))

    # Strat de pooling, folosit pentru a reduce dimensiunile volumului de output
    # Dandu-se un kernel de 2 x 2, se va calcula maximul minorilor de ordin 2
    # din matricea de ponderi, pastrandu-se astfel cele mai frecvente
    # caracteristici
    # Aceasta tehnica ajuta la pastrarea invarianta a modelului in cazul
    # distorsionarii usoare a datelor de antrenare
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
30
31      # Crestem numarul de filtre pentru a putea permite invatarea unor
32      # caracteristici mai complexe
33      model.add(Conv2D(128, (5, 5), activation='relu', kernel_regularizer=tf.keras.
        regularizers.l2(0.001)))
34
35      # Strat de pooling, folosit pentru a reduce dimensiunile volumului de output
36      model.add(MaxPooling2D(pool_size=(2, 2)))
37
38      # Acest strat reduce matricea de imagini la un vector unidimensional
39      model.add(Flatten())
40
41      # Strat complet conectat (fiecare neuron primeste input de la toti neuronii din
42      # stratul precedent)
43      model.add(Dense(750, activation='relu', kernel_regularizer=tf.keras.regularizers
        .l2(0.001)))
44
45      # Se renunta in mod aleator la 50% dintre caracteristicile invatate pana acum
46      model.add(Dropout(0.5))
47
48      # Strat complet conectat (fiecare neuron primeste input de la toti neuronii din
49      # stratul precedent)
50      model.add(Dense(250, activation='relu', kernel_regularizer=tf.keras.regularizers
        .l2(0.001)))
51
52      # Strat de output, imaginile vor fi distribuite in 9 clase
53      # Functia de activare transforma un vector de valori reale intr-un vector cu
        valori a caror suma
54      # este 1, fiecare valoare fiind comprimata in intervalul [0, 1) asa incat sa
        poata fi
55      # interpretate drept probabilitati
56      model.add(Dense(9, activation='softmax'))
57
58      # Functia de pierdere: specifica problemelor de multi-clasificare (mai mult de 2
         etichete)
59      # Are rolul de a furniza cate o probabilitate pentru fiecare din cele 9 etichete
         finale
60      # pentru o imagine
61      # Functia de optimizare simuleaza algoritmul de scadere dupa gradientul
        stochastic
62      model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['
        accuracy'])
63      # model.summary()
64      """
65      Model: "sequential"
66      _________________________________________________________________
67      Layer (type)                 Output Shape              Param #
68      =================================================================
69      conv2d (Conv2D)              (None, 28, 28, 32)        832
70      _________________________________________________________________
71      max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
72      _________________________________________________________________
73      conv2d_1 (Conv2D)            (None, 10, 10, 128)       102528
74      _________________________________________________________________
75      max_pooling2d_1 (MaxPooling2 (None, 5, 5, 128)         0
76      _________________________________________________________________
77      flatten (Flatten)            (None, 3200)              0
78      _________________________________________________________________
```

```
    dense (Dense)                (None, 750)               2400750
    _________________________________________________________________
    dropout (Dropout)            (None, 750)               0
    _________________________________________________________________
    dense_1 (Dense)              (None, 250)               187750
    _________________________________________________________________
    dense_2 (Dense)              (None, 9)                 2259
    =================================================================
    Total params: 2,694,119
    Trainable params: 2,694,119
    Non-trainable params: 0
    _________________________________________________________________
    """

    return model
```

Legea funcției ReLU: h = max(0, a) unde a = W$\mathbf{x}$ + b, funcție liniară, cost redus la utilizare. Avantaje:

- Performanță mai bună la convergență. Gradul redus de dispariție al gradientului apare atunci când a > 0. În acest caz, gradientul are o valoare constantă, ceea ce duce la o învățare mai rapidă
- Gradul de împrăștiere care apare atunci când a ≤ 0. Cu cât există mai multe unități din acestea într-un strat, cu atât reprezentarea rezultată va fi mai "împrăștiată", ceea ce par a fi mai benefice decât reprezentările dense

Legea stratului dens: output = activation(dot(input, kernel) + bias)

- bias = eroare sistematică, denotă underfit; corectată prin creșterea complexității modelului

Despre Dropout:

- neuronii sunt încurajați să învețe feature-uri utile fără să se bazeze pe alți neuroni
- după antrenarea modelului, întreaga rețea e folosită pentru inferență

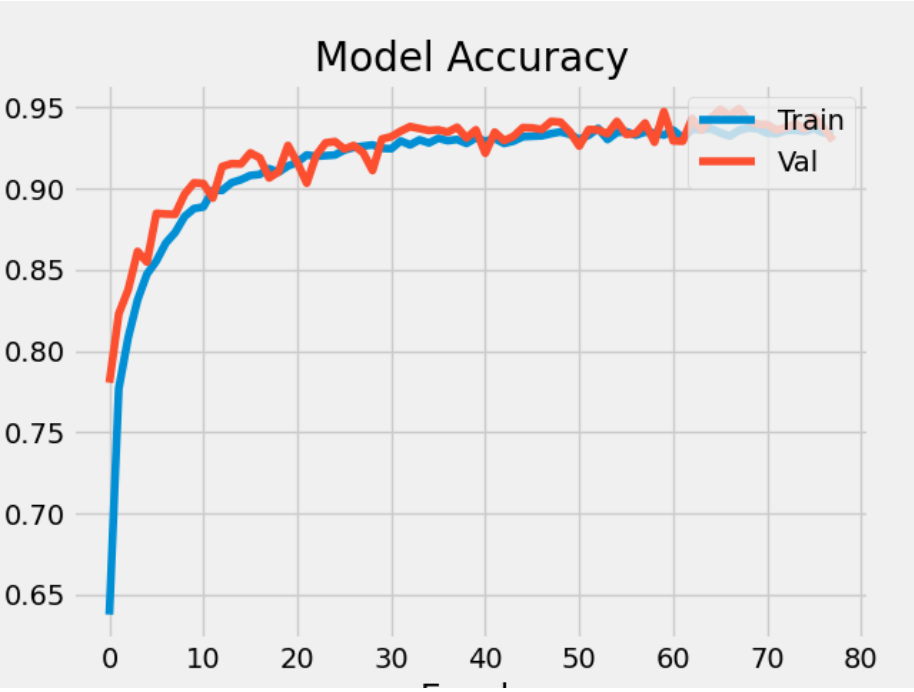

Figure 1: Acuratețea celui mai bun model de pe Kaggle

Figure 2: Funcția de pierdere a celui mai bun model de pe Kaggle

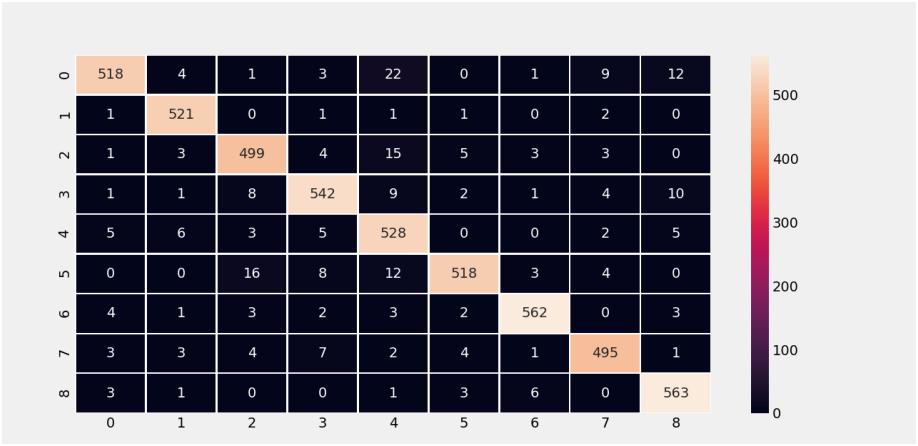

Figure 3: Matricea de confuzie a celui mai bun model de pe Kaggle