مقایسه روش های مختلف طبقه بندی روی یک مسئله Benchmark
گزارش پروژه ۷ هوش مصنوعی دکتر قطعی

امین رضائی
دانشجوی کارشناسی علوم کامپیوتر دانشگاه صنعتی امیرکبیر

۶ خرداد ۱۴۰۰

چکیده

در این پروژه قصد داریم روش های مختلف طبقه بندی را از جمله مدل های رگرسیونی و شبکه های عصبی مختلف را روی یک مسئله بنچمارک معروف اعمال و تست کنیم. مسئله بنچمارک انتخاب شده در این پروژه طبقه بندی ارقام مجموعه دادگان MNIST خواهد بود.

۱ مقدمه

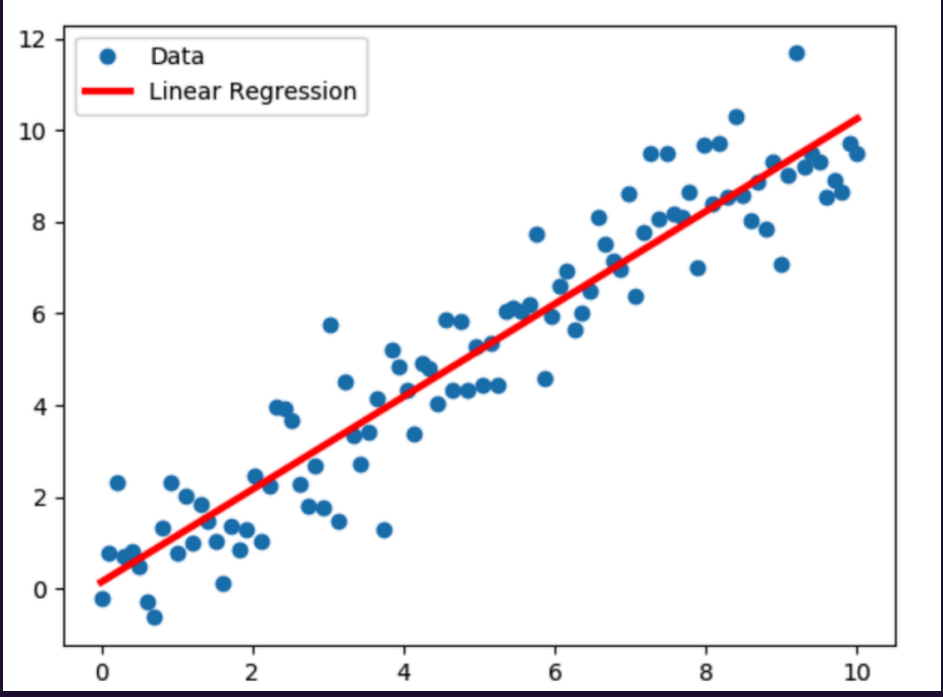
در این پروژه قصد داریم روش های مختلف طبقه بندی را بر روی مجموعه دادگان ارقام دست نویس MNIST اجرا و از مقایسه آنها گزارشی تهیه کنیم . مدلهای مورد استفاده مدلهای رگرسیونی مختلف و مدلهای شبکه عصبی مختلف خواهند بود. از کتابخانه Keras برای دریافت کردن دیتاست ارقام و همچنین ساختن مدلهای شبکه عصبی ، از Scikit-Learn برای ساختن مدلهای رگرسیونی و از کتابخانه های Matplotlib و Seaborn برای نمایش و ویژوالایز کردن داده ها استفاده خواهد شد [۳، ۴، ۵، ۶، ۷]

۲ مدلهای استفاده شده

۱.۲ مدل رگرسیون خطی (Linear Regression)[۸]

این مدل سعی میکند بصورت خطی داده ها را از هم جدا کند و در پی یافتن ضرایب $w = (w_1, w_2, \ldots, w_p)$ برای منیمم سازی مجموع خطای مربعات بین داده های مشاهده شده و کلاس های تخمین زده شده است. همانطور که از اسمش پیداست این مدل توانایی شناسایی و تخمین توزیع های غیرخطی را ندارد.

این مدل را میتوان با یک شبکه عصبی که یک لایه ورودی و یک لایه خروجی دارد و از تابع فعالساز خطی استفاده میکند مدلسازی و حل کرد.

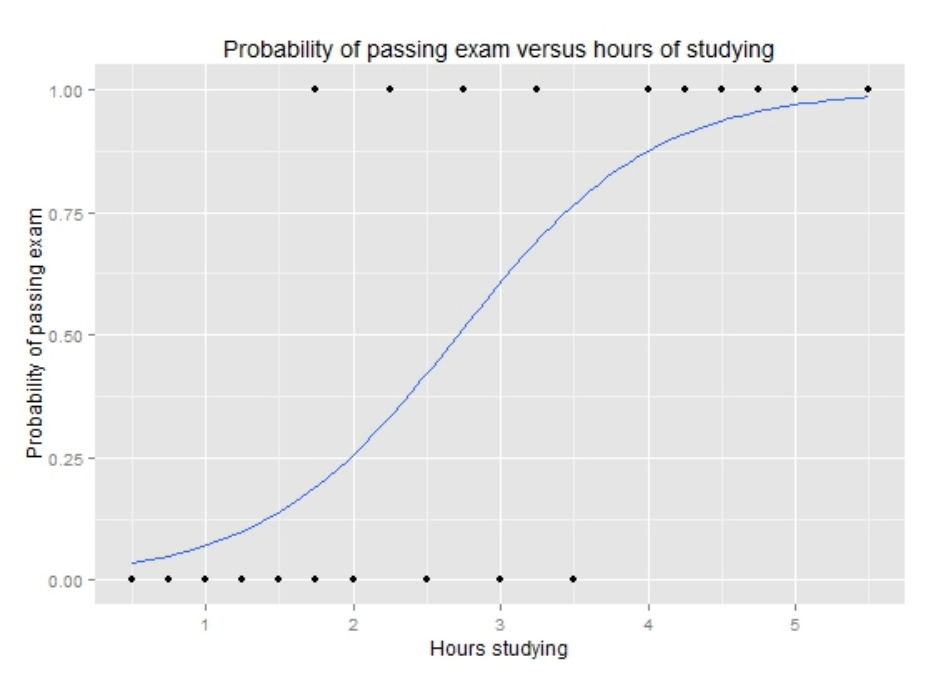


شکل ۱: نمونه رگرسیون خطی برای داده های دوبعدی

۲.۲ مدل رگرسیون لوجستیگ (Logitistic Regression)[۹، ۱۰]

در این مدل رگرسیون یک تابع لوجیت (سیگموید و ...) روی حاصل ترکیب خطی وزن ها و مقادیر ورودی اعمال میشود که نقش تابع احتمال را بازی میکند. این مدل بیشتر برای کلاس بندی استفاده میشود و از Maxiumum Likelihood برای بهینه سازی و کاهش خطا استفاده میکند. این مدل بدلیل ساختار غیرخطی تابع لوجیت توانایی تشخیص توزیع های غیرخطی را نیز داراست.

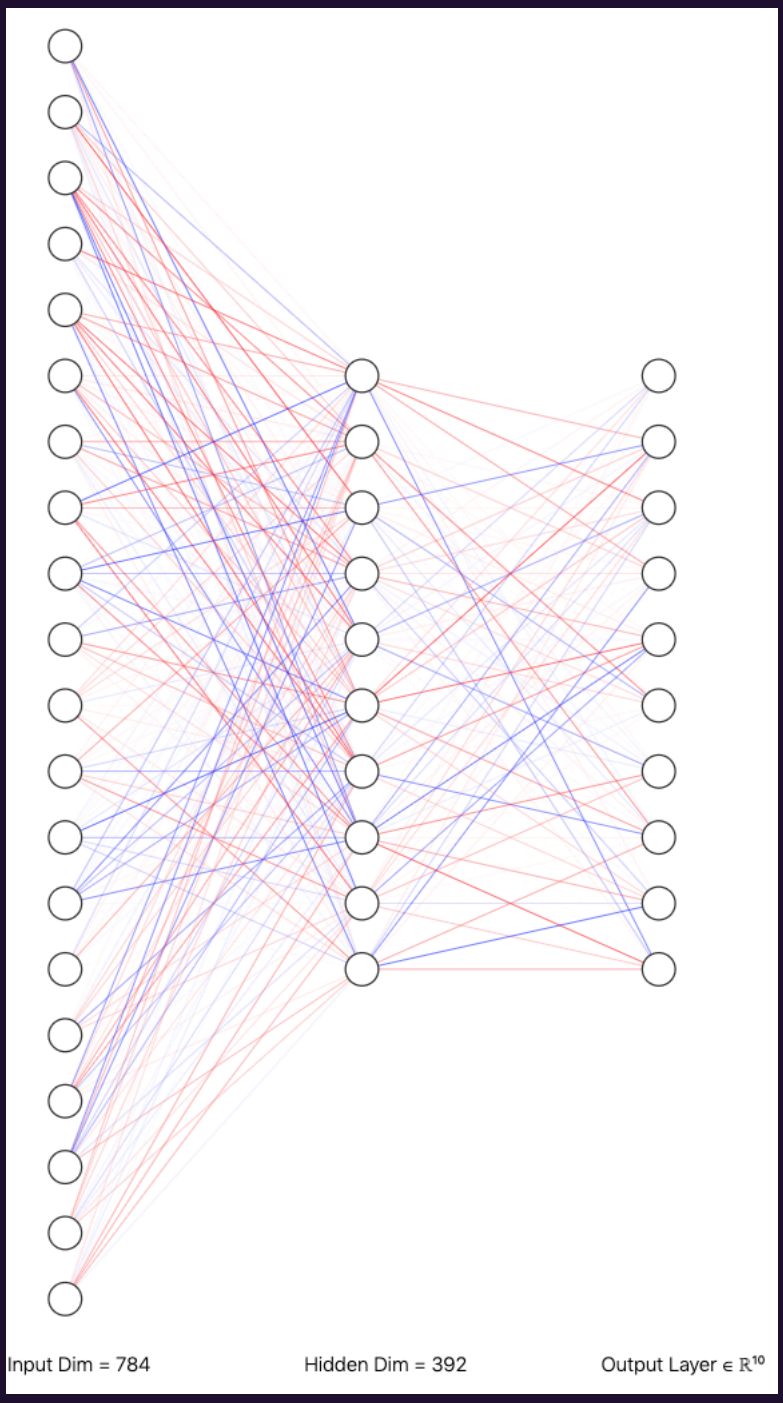
این مدل را میتوان با یک شبکه عصبی که یک لایه ورودی و یک لایه خروجی دارد و از تابع فعالساز لوجیت مانند سیگموید استفاده میکند مدلسازی و حل کرد.



شکل ۲: نمونه رگرسیون لوجستیک

۳.۲ شبکه عصبی MLP با یک لایه مخفی

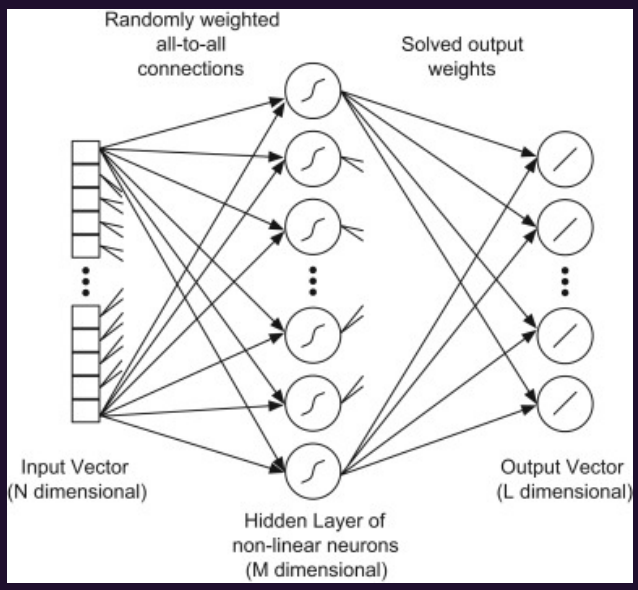
در این مدل از شبکه عصبی ای استفاده خواهیم کرد که پیکسل های تصویر را در لایه ورودی گرفته و لایه مخفی ای به اندازه ۳۹۲ دارد و بعد خروجی اش ۱۰ است. دسته داده ها بصورت کتگوریکال توسط One-Hot Encoding به داده ۱۰ بعدی انکد میشوند. برای لایه مخفی از تابع اکتیویشن ReLU و برای لایه خروجی از تابع اکتیویشن Softmax استفاده خواهد شد ساختار شبکه بصورت زیر است:



شکل ۳: ساختار شبکه

۴.۲ شبکه عصبی ELM

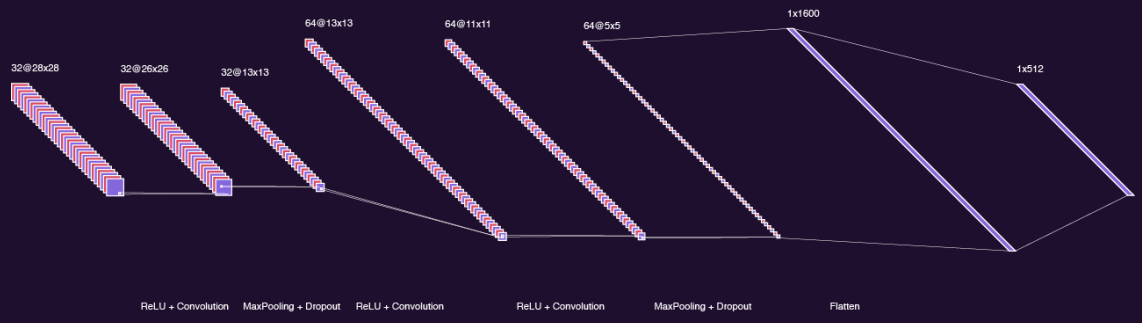
این نوع شبکه عصبی که قبلا ساختارش را بررسی کرده ایم، نوعی شبکه عصبی feedforward است که اتصالات ورودی به لایه مخفی به صورت رندوم وزن دهی میشوند و درطول آموزش ثابت هستند و تنها اتصالات لایه مخفی به لایه خروجی آموزش داده میشوند.



شکل ۴: ساختار شبکه های ELM [۱۱]

۵.۲ شبکه عصبی کانولوشنال [۱۲، ۱۳]

در این نوع شبکه های عصبی عمیق که اغلب برای تسک های پردازش تصویر و ورودی های تصویری استفاده میشوند، از مفهومی به نام کانولوشن که نوعی تبدیل خطی است استفاده میشود بصورتی که ماتریس هایی تحت عنوان فیلتر در قسمت های مختلف تصویر ضرب میشوند و فیچر و ویژگی ای درمورد آن ناحیه استخراج میکنند. همچنین لایه هایی تحت عنوان Max Pooling شاخص ترین ویژگی ها را از بین ویژگی های زیادی که کانولوشن تولید کرده است را انتخاب میکنند. ساختار های متفاوتی برای این نوع شبکه ها موجود است که تعداد متعدد لایه های کانولوشن یا Max Pooling را دارند ،این نوع شبکه ها توان پردازشی و زمان پردازش بالایی را نیاز دارند و معمولا با GPU های قوی آموزش داده میشوند. معماری ای که در این پروژه بررسی خواهیم کرد بصورت زیر است:



شکل ۵: ساختار شبکه CNN

۳ پیاده سازی

ایمپورت کتابخانه های مورد نیاز:

[1]:
```
from keras.datasets import mnist
from keras.utils import to_categorical
import numpy as np
import tensorflow as tf
import tqdm
import seaborn as sns
import matplotlib.pyplot as plt
```

۱.۳ پیش پردازش داده ها

داده های MNIST را توسط Keras لود میکنیم که به صورت پیکسل هایی در قالب ارایه های ۲۸ در ۲۸ هستند و مقداری بین ۰ و ۲۵۵ دارند. داده ها را که بصورت ۲۸ در ۲۸ هستند را تغییر شکل میدهیم تا به شکل ارایه هایی بطول ۷۸۴ بشوند. برای اینکه داده ها بین ۰ و ۱ باشند و پردازش و یادگیری برای شبکه اسانتر باشد تمام داده هارا بر ۲۵۵ تقسیم میکنیم. همچنین کلاس های تصاویر را که بصورت یک عدد است را توسط تابع to_categorical کتابخانه کراس بصورت One-Hot انکد میکنیم

[2]:
```
input_dim = 28 * 28
classes = 10
(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(-1, input_dim) / 255.
X_test = X_test.reshape(-1, input_dim) / 255.
X_train = X_train.astype(np.float32)
X_test = X_test.astype(np.float32)
y_train_not_encoded = y_train
y_test_not_encoded = y_test
y_train = to_categorical(y_train, num_classes=classes)
y_test = to_categorical(y_test, num_classes=classes)
y_train = y_train.astype(np.float32)
y_test = y_test.astype(np.float32)
```

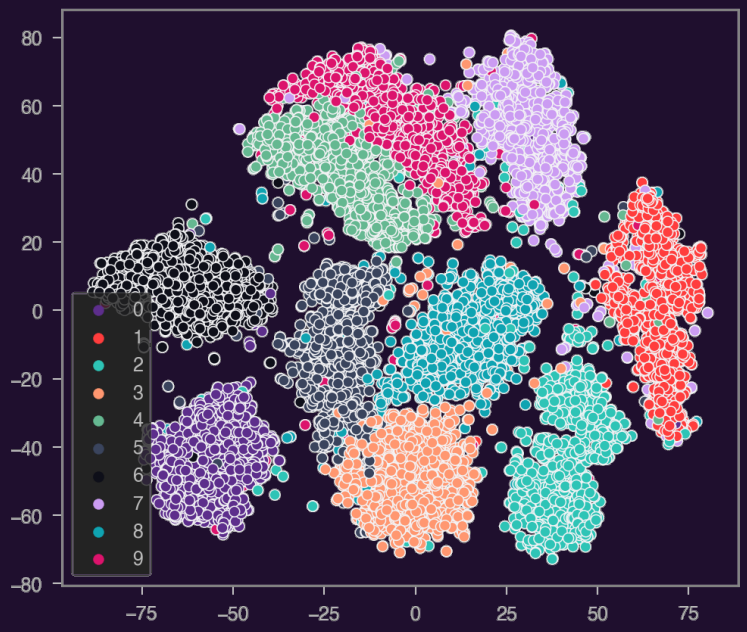
```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 27s 2us/step
```

داده هایمان را که به بردار های ۷۸۴ بعدی هستند را برای اینکه بتوانیم توزیعشان را در فضای دو بعدی مشاهده کنیم و قادر به ویژوالایز کردن باشیم توسط TSNE [۱۴] به ۲ بعد کاهش میدهیم.

[3]:
```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2)
embed_test = tsne.fit_transform(X_test)
```

[4]:
```
colors = [
"#5D2E8C","#FF3D3D","#2EC4B6","#FF9770","#65B891",
"#3A445D","#0E0F19","#CB9CF2","#0FA3B1","#DC136C"]
palette = sns.color_palette(colors)
f, ax = plt.subplots(1,1, dpi=144)
sns.scatterplot(x=embed_test[:, 0], y=embed_test[:, 1],
hue=y_test_not_encoded,
style=y_test_not_encoded,
ax=ax,
palette=palette,
edgecolor="#F4F4F6",
markers=["o" for _ in range(10)])
```

توزیع داده ها به همراه کلاس هایشان بصورت زیر است:



۲.۳ مدل رگرسیون خطی

مسئله را با رگرسیون خطی و توسط کلاس LinearRegression از کتابخانه Scikit-Learn حل میکنیم و درصد دقت را بدست میاوردیم

[5]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score

regressor = LinearRegression()
regressor.fit(X_train, y_train_not_encoded)
y_pred = regressor.predict(X_test)
y_pred = np.round(y_pred).astype(np.int)
accuracy_score(y_pred, y_test_not_encoded)
```

[5]:
```
0.2202
```

دقت این مدل 22% است که بسیار کم است، همانگونه که توزیع داده هارا مشاهده کردیم الگوی داده ما خطی نیس و رگرسیون خطی نیز از تشخیص و کلاس بندی این مدل ناتوان ماند. پس باید به سمت مدل های بهتری برویم

۳.۳ مدل رگرسیون لوجستیک

مسئله را با رگرسیون لوجستیک و توسط کلاس LogisticRegression از کتابخانه Scikit-Learn حل میکنیم و درصد دقت را بدست میاوردیم

[6]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

regressor = LogisticRegression()
regressor.fit(X_train, y_train_not_encoded)
y_pred = regressor.predict(X_test)
accuracy_score(y_pred, y_test_not_encoded)
```

[6]:
```
0.9256
```

دقت به دست امده خیلی خوب است و مدل توانسته است 92.56% درصد دسته بندی ها را به درستی پاسخ دهد به سراغ مدل های شبکه عصبی میرویم تا دقت و عملکرد انها را مشاهده کنیم

۴.۳ مدل شبکه عصبی MLP با یک لایه مخفی

مدلمان را با سه لایه توسط keras میسازیم که دارای ۷۸۴ نورون در لایه ورودی، ۳۹۲ نورون در لایه مخفی و ۱۰ نورون در لایه خروجی دارد و از اکتیویشن ReLU برای لایه هیدن و اکتیویشن Softmax برای لایه خروجی استفاده میکند

[7]:
```
from tensorflow.python.util import deprecation
deprecation._PRINT_DEPRECATION_WARNINGS = False
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(392, input_dim=input_dim, activation="relu"))
model.add(Dense(10, activation="softmax"))

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

[8]:
```
model.fit(X_train, y_train, batch_size=32, epochs=10, validation_split=0.1)
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
loss: 0.2200 - acc: 0.9357 - val_loss: 0.1066 - val_acc: 0.9695
Epoch 2/10
loss: 0.0892 - acc: 0.9732 - val_loss: 0.0874 - val_acc: 0.9735
Epoch 3/10
loss: 0.0576 - acc: 0.9817 - val_loss: 0.0642 - val_acc: 0.9802
Epoch 4/10
loss: 0.0399 - acc: 0.9874 - val_loss: 0.0776 - val_acc: 0.9788
Epoch 5/10
loss: 0.0296 - acc: 0.9906 - val_loss: 0.0753 - val_acc: 0.9812
Epoch 6/10
loss: 0.0233 - acc: 0.9928 - val_loss: 0.0676 - val_acc: 0.9828
Epoch 7/10
loss: 0.0179 - acc: 0.9941 - val_loss: 0.0839 - val_acc: 0.9787
Epoch 8/10
loss: 0.0154 - acc: 0.9950 - val_loss: 0.1083 - val_acc: 0.9758
Epoch 9/10
```

```
loss: 0.0120 - acc: 0.9960 - val_loss: 0.0797 - val_acc: 0.9803
Epoch 10/10
loss: 0.0122 - acc: 0.9959 - val_loss: 0.0932 - val_acc: 0.9812
```

[9]:
```
model.evaluate(X_test, y_test, batch_size=32)
```

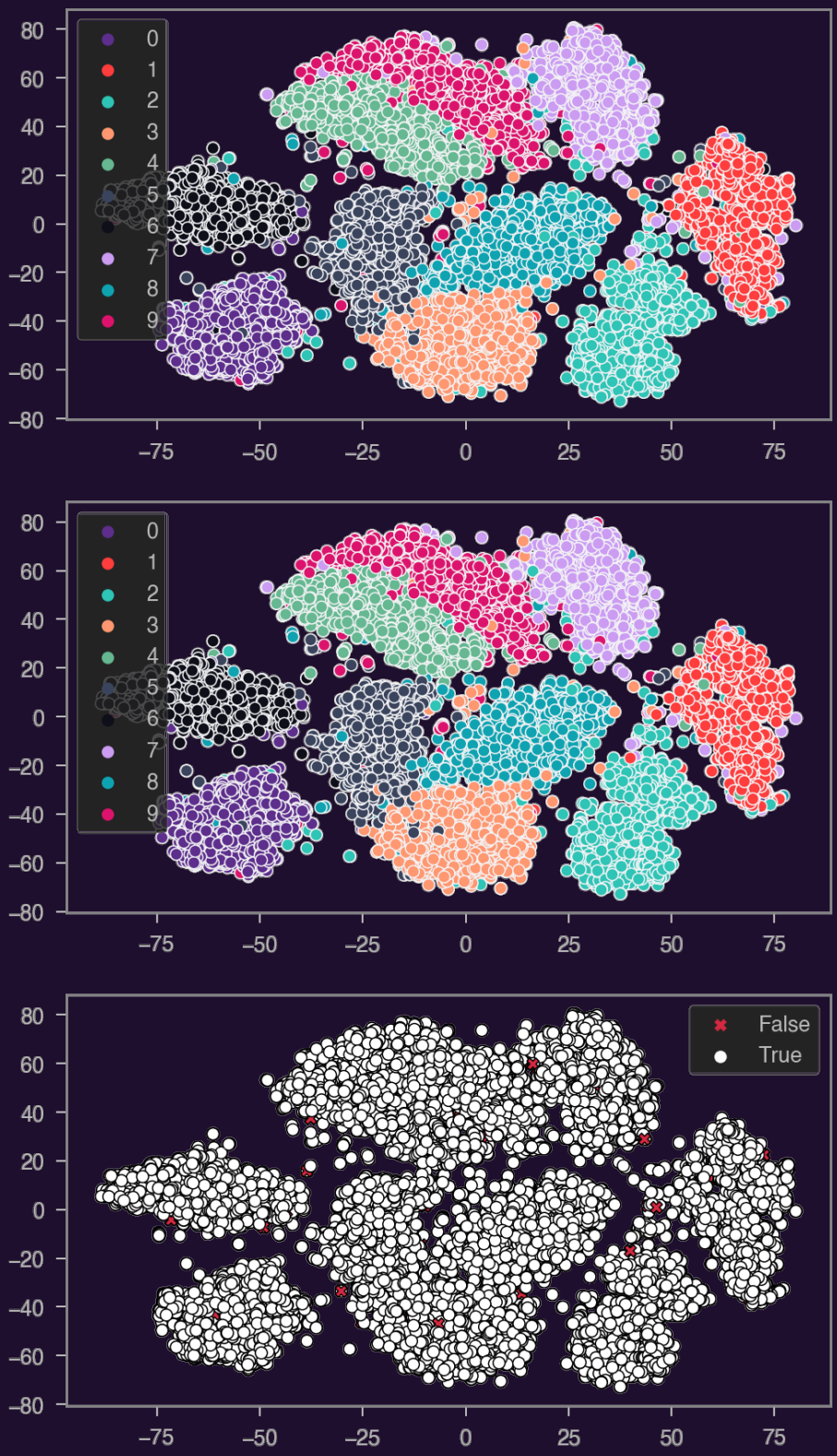
[9]:
```
10000/10000 - 0s 49us/sample - loss: 0.0775 - acc: 0.9810
[0.07746978181417453, 0.981]
```

مدل شبکه عصبیمان توانست به دقت 98.1% برسد که دقت خیلی بالاتری از مدل رگرسیون لوجستیک است در قسمت زیر داده های پیشبینی شده را در قالب نمودار نمایش میدهیم و موارد شناسایی شده را درست و غلط بودنشان رو نشان میدهیم.

[10]:
```
y_pred = model.predict_classes(X_test, batch_size=32)
```

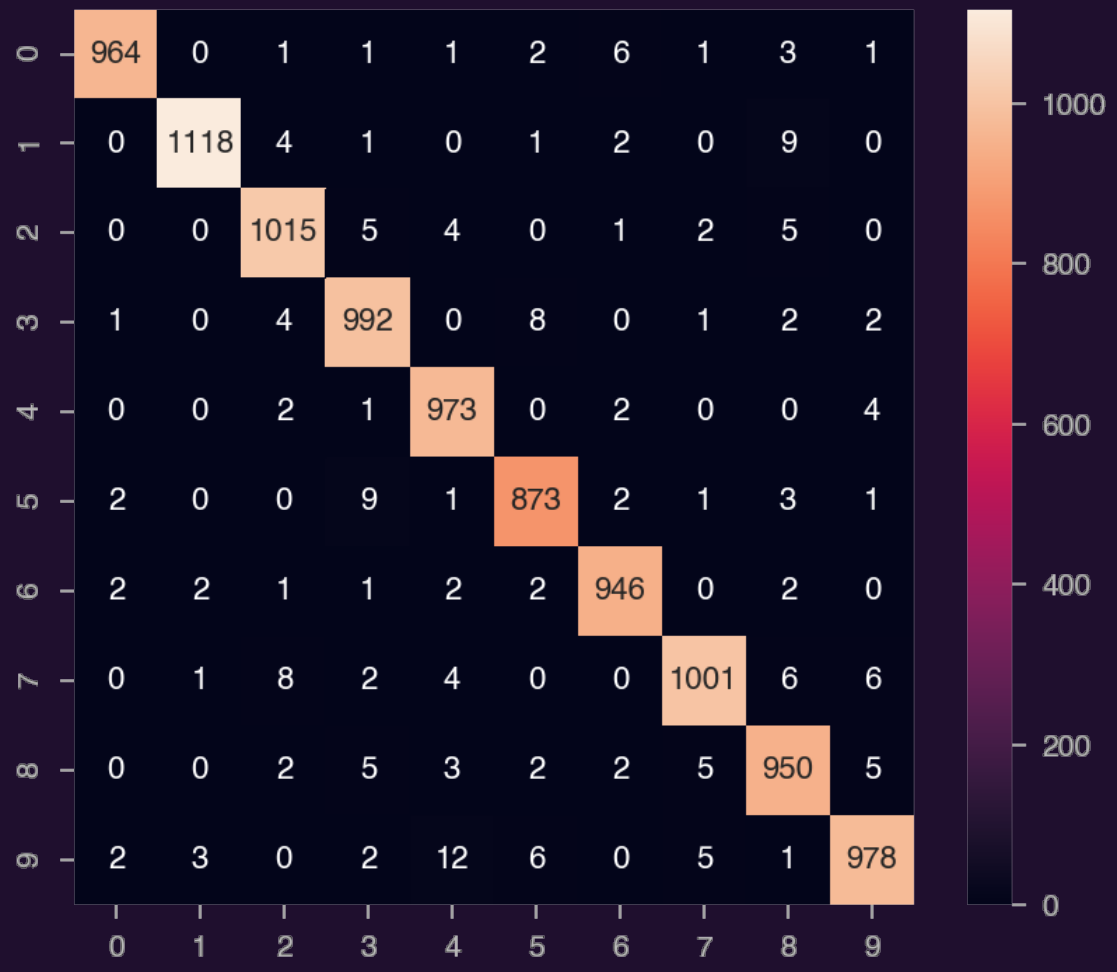
[11]:
```
f, ax = plt.subplots(3,1,figsize=(8,15), dpi=144)
pal_tr = sns.color_palette(["#D52941","#ffffff"])
sns.scatterplot(x=embed_test[:, 0], y=embed_test[:, 1],
                ax=ax[0],
                hue= y_test_not_encoded,
                style= y_test_not_encoded,
                palette=palette,
                edgecolor="#F4F4F6",
                markers=["o" for _ in range(10)])
sns.scatterplot(x=embed_test[:, 0], y=embed_test[:, 1],
                ax=ax[1],
                hue=y_pred,
                style=y_pred,
                palette=palette,
                edgecolor="#F4F4F6",
                markers=["o" for _ in range(10)])
sns.scatterplot(x=embed_test[:, 0], y=embed_test[:, 1],
                ax=ax[2],
                hue=y_pred == y_test_not_encoded,
                style=y_pred == y_test_not_encoded,
                palette=pal_tr,edgecolor="#000000",
                markers=["X","o"])
```

نمودار اول توزیع خود داده های تست، نمودار دوم توزیع داده های تست با لیبل های پیشبینی شده و نمودار اخر کیس هایی که درست پیشبینی شده اند با نقطه های سفید و کیس های نادرست با علامت ضربدر قرمز نمایش داده شده اند.



[12]:
```
from sklearn.metrics import confusion_matrix
f, ax = plt.subplots(1,1, dpi=121)
mat = confusion_matrix(y_test_not_encoded, y_pred)
sns.heatmap(mat, annot=True, fmt="d", ax=ax)
```

همچنین توسط توابع کتابخانه sklearn به محاسبه Confusion Matrix [۱۵] میپردازیم که پرفورمنس و عملکرد مدل ما را در دسته بندی کلاس های مختلف دراختیار میگذارد و سپس آنرا بصورت نموداری نمایش میدهیم



۵.۳ شبکه ELM

از کتابخانه OS-ELM برای پیاده سازی این شبکه استفاده میکنیم . این شبکه سه لایه خواهد داشت که لایه ورودی و خروجی مانند مدل قبل به ترتیب ۷۸۴ و ۱۰ نورون خواهند داشت و لایه مخفی ۷۰۰۰ نورون خواهد داشت چون درلایه اول وزن ها به تصادف تعیین میشوند و یادگیری در ان لایه رخ نمیدهد به تعداد بیشتری نورون از حالت یادگیری backpropagation نیاز داریم

[13]:
```
!git clone https://github.com/otenim/TensorFlow-OS-ELM
!cp TensorFlow-OS-ELM/os_elm.py ./
```

[14]:
```
from os_elm import OS_ELM
```

[15]:
```
def softmax(a):
    c = np.max(a, axis=-1).reshape(-1, 1)
    exp_a = np.exp(a - c)
    sum_exp_a = np.sum(exp_a, axis=-1).reshape(-1, 1)
    return exp_a / sum_exp_a
```

[16]:
```
hidden_dim = 7000
elm = OS_ELM(
        n_input_nodes= input_dim,
        n_hidden_nodes= hidden_dim,
        n_output_nodes= classes,
        loss='categorical_crossentropy',
        activation='sigmoid')

border = int(1.5 * hidden_dim)
X_train_init = X_train[:border]
X_train_seq = X_train[border:]
y_train_init = y_train[:border]
y_train_seq = y_train[border:]
```

[17]:
```
pbar = tqdm.tqdm(total=len(X_train), desc='initial training phase')
elm.init_train(X_train_init, y_train_init)
pbar.update(n=len(X_train_init))
```

```
    pbar.set_description('sequential training phase')
    batch_size = 64
    for i in range(0, len(X_train_seq), batch_size):
        X_batch = X_train_seq[i:i+batch_size]
        y_batch = y_train_seq[i:i+batch_size]
        elm.seq_train(X_batch, y_batch)
        pbar.update(n=len(X_batch))
    pbar.close()
```

[18]:
```
    res = elm._OS_ELM__sess.run([elm._OS_ELM__loss,elm._OS_ELM__accuracy],␣
↪feed_dict={elm._OS_ELM__x: X_test, elm._OS_ELM__t: y_test})
    print("accuracy %f" % res[1])
```

```
accuracy 0.972400
```

دقتی که این شبکه توانست به دست بیاورد 97.24% است همچنان از مدل رگرسیونی تفاوت زیادی دارد اما نتوانست دقت بهتری از مدل MLP که بر پایه یادگیری BackPropagtion بود را بدست بیاورد.

۶.۳ شبکه عصبی کانولوشنال (CNN)

ابتدا داده های ورودی را به طور ۴ بعدی تغییر میدهیم که به شکل

$$(Number\ Of\ Images, Image\ Width, Image\ Height, Depth)$$

خواهند شد و سپس شبکه را توسط keras با معماری معرفی شده در قسمت قبل میسازیم. برای لایه های میانی از اکتیویشن ReLU و برای اکتیویشن خروجی از Softmax استفاده میشود.

[19]:
```
    import keras
    from keras.models import Sequential
    from keras.layers import Dense, Dropout, Activation, Flatten
    from keras.layers import Conv2D, MaxPooling2D
    from keras import backend as k

    img_dim = (28, 28)
    if k.image_data_format() == 'channels_first':
        X_train = X_train.reshape(X_train.shape[0], 1, img_dim[0], img_dim[1])
        X_test = X_test.reshape(X_test.shape[0], 1, img_dim[0], img_dim[1])
        input_shape = (1, img_dim[0], img_dim[1])
    else:
```

```
    X_train = X_train.reshape(X_train.shape[0], img_dim[0], img_dim[1], 1)
    X_test = X_test.reshape(X_test.shape[0], img_dim[0], img_dim[1], 1)
    input_shape = (img_dim[0], img_dim[1], 1)

model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=input_shape))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(classes))
model.add(Activation('softmax'))
```

[20]:
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

[21]:
```
model.fit(X_train, y_train, batch_size=256, epochs=15, validation_split=0.1)
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/15
loss: 0.3186 - acc: 0.8976 - val_loss: 0.0494 - val_acc: 0.9865
Epoch 2/15
```

```
loss: 0.0759 - acc: 0.9765 - val_loss: 0.0383 - val_acc: 0.9888
Epoch 3/15
loss: 0.0556 - acc: 0.9830 - val_loss: 0.0319 - val_acc: 0.9900
Epoch 4/15
loss: 0.0411 - acc: 0.9871 - val_loss: 0.0325 - val_acc: 0.9907
Epoch 5/15
loss: 0.0370 - acc: 0.9884 - val_loss: 0.0256 - val_acc: 0.9930
Epoch 6/15
loss: 0.0316 - acc: 0.9903 - val_loss: 0.0273 - val_acc: 0.9928
Epoch 7/15
loss: 0.0273 - acc: 0.9914 - val_loss: 0.0308 - val_acc: 0.9918
Epoch 8/15
loss: 0.0250 - acc: 0.9921 - val_loss: 0.0237 - val_acc: 0.9940
Epoch 9/15
loss: 0.0218 - acc: 0.9928 - val_loss: 0.0283 - val_acc: 0.9930
Epoch 10/15
loss: 0.0214 - acc: 0.9934 - val_loss: 0.0234 - val_acc: 0.9935
Epoch 11/15
loss: 0.0184 - acc: 0.9938 - val_loss: 0.0228 - val_acc: 0.9935
Epoch 12/15
loss: 0.0180 - acc: 0.9943 - val_loss: 0.0192 - val_acc: 0.9943
Epoch 13/15
loss: 0.0161 - acc: 0.9946 - val_loss: 0.0225 - val_acc: 0.9948
Epoch 14/15
loss: 0.0150 - acc: 0.9950 - val_loss: 0.0215 - val_acc: 0.9947
Epoch 15/15
loss: 0.0163 - acc: 0.9942 - val_loss: 0.0244 - val_acc: 0.9945
```

[22]:
```
model.evaluate(X_test, y_test, batch_size=256)
```

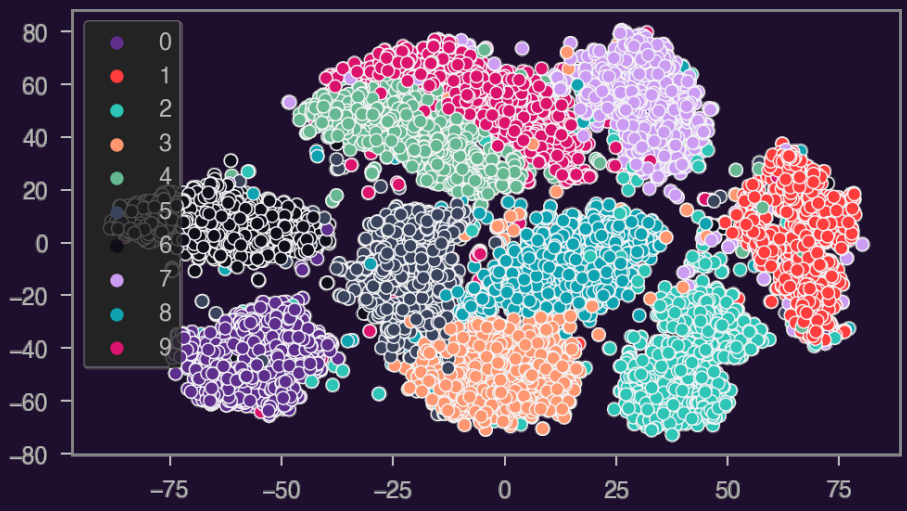
[22]:
```
10000/10000 [==============================] - 0s 18us/step
[0.01706054805782187, 0.9947]
```

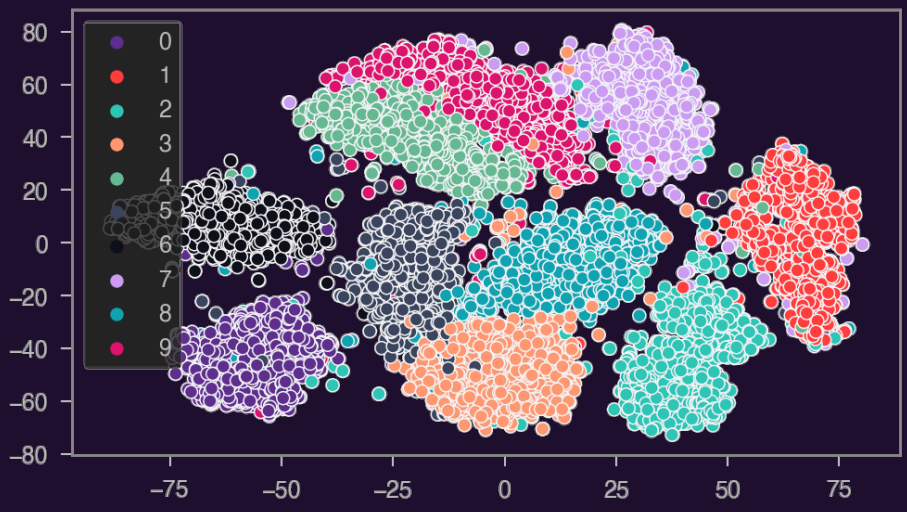
دقتی که شبکه CNN توانست به دست بیاورد 99.47% است و توسط این مدل توانستیم به دقت قابل توجه بالاتری برسیم . فرآیند یادگیری خوب بوده است و بسیاری از موارد تست را توانسته به درستی تشخیص دهد.

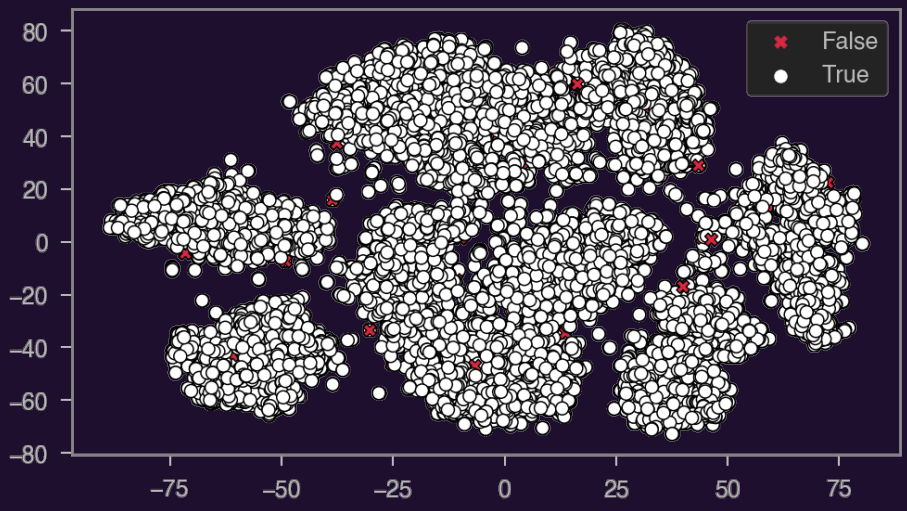
[23]:
```
y_pred = model.predict_classes(X_test, batch_size=256)
```

[24]:
```
f, ax = plt.subplots(3,1,figsize=(8,15), dpi=144)
pal_tr = sns.color_palette(["#D52941","#ffffff"])
sns.scatterplot(x=embed_test[:, 0], y=embed_test[:, 1],
                ax=ax[0],
                hue= y_test_not_encoded,
                style= y_test_not_encoded,
                palette=palette,
                edgecolor="#F4F4F6",
                markers=["o" for _ in range(10)])
sns.scatterplot(x=embed_test[:, 0], y=embed_test[:, 1],
                ax=ax[1],
                hue=y_pred,
                style=y_pred,
                palette=palette,
                edgecolor="#F4F4F6",
                markers=["o" for _ in range(10)])
sns.scatterplot(x=embed_test[:, 0], y=embed_test[:, 1],
                ax=ax[2],
                hue=y_pred == y_test_not_encoded,
                style=y_pred == y_test_not_encoded,
                palette=pal_tr,
                edgecolor="#000000",
                markers=["X","o"])
```

همانند شبکه قبل برای این شبکه نیز نمودار ها را رسم میکنیم. همانطور که انتظار داریم نقاط قرمز در نمودار اخر کاهش قابل توجهی را داشته اند که حاصل استفاده از مدل بهتر با دقت بالاتر است.

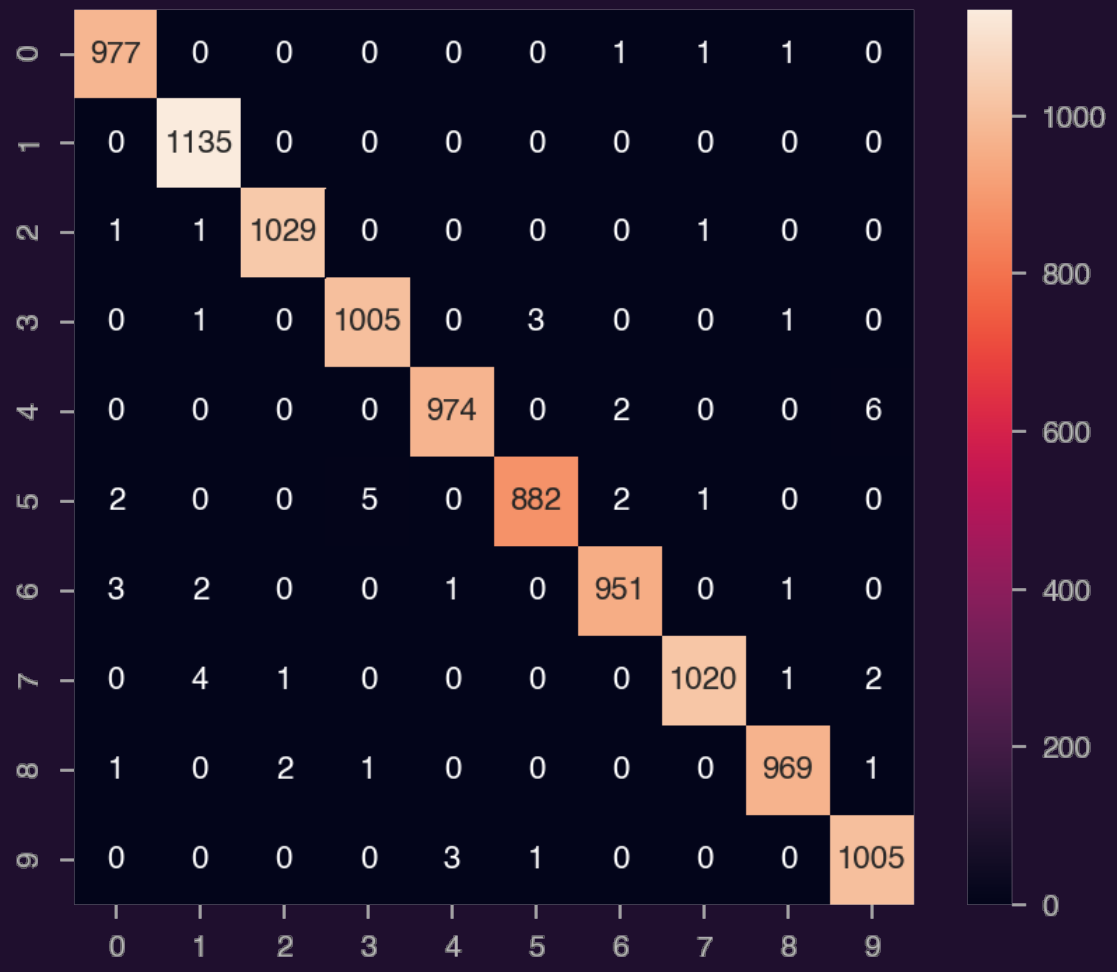

0
1
2
3
4
5
6
7
8
9
80
60
40
20
0
−20
−40
−60
−80
−75
−50
−25
0
25
50
75


0
1
2
3
4
5
6
7
8
9
80
60
40
20
0
−20
−40
−60
−80
−75
−50
−25
0
25
50
75


False
True
80
60
40
20
0
−20
−40
−60
−80
−75
−50
−25
0
25
50
75

[25]:
```
from sklearn.metrics import confusion_matrix
f, ax = plt.subplots(1,1, dpi=121)
mat = confusion_matrix(y_test_not_encoded, y_pred)
sns.heatmap(mat, annot=True, fmt="d", ax=ax)
```

برای این مدل نیز Confusion Matrix را بصورت نمودار نمایش میدهیم. همانطور که مشاهده میشود اعداد خارج قطر اصلی مقدار اکثرشان صفر و درصورت صفر نبودن مقادیر کوچکیست.



۴ نتیجه گیری

با انجام این پروژه و مقایسه دقت های مدلهای مختلف نتیجه میگیریم که مدلهای شبکه عصبی که دارای لایه پنهان نیز هستند درمقایسه با مدلهای رگرسیونی از دقت بسیار بالاتری برخوردار هستند و توانایی و قدرت بالاتری از مدلهای رگرسیونی در مسائل کلاس بندی و پیشبینی دارند. در بین شبکه های عصبی نیز در این مورد دیدیم که شبکه عصبی کانولوشنال خیلی بهتر از بقیه مدلهای شبکه در دسته بندی عمل کرد و بالاترین دقت را بدست اورد. درکل مدلهای طبقه بندی با انواع مختلف خود قدرت خوبی در اجرای مسائل طبقه بندی دارند و شبکه های عصبی در طی چندسال اخیر توانسته اند مسائل مختلفی را حل کنند و ارتباط بین داده ها را به خوبی یاد بگیرند. با انتخاب مدل های درست برای مسئله میتوان سیستم های هوشمند را توسعه داد و از آنها بهره گرفت.

جدول مقایسه دقت مدل ها

Model	Accuracy
Linear Regression	22.02%
Logistic Regression	92.56%
Neural Network (Single Hidden Layer)	98.1%
Online Sequential ELM	97.24%
Convolutional Neural Network	**99.47%**

کد و پیاده سازی کامل کد این پروژه در رپوزیتوری گیتهاب که از لینک زیر در دسترس است موجود است:

https://github.com/AminRezaei0x443/minst_classification

References

[1] M. Ghatee. Artificial intelligence lecture and slides.

[2] M. Ghatee. Neural network lecture and slides.

[3] Keras Framework and Datasets. https://keras.io/.

[4] Matplotlib Plotting Library. https://matplotlib.org/.

[5] Seaborn: Statistical Data Visualization. https://seaborn.pydata.org/.

[6] Numpy. https://github.com/numpy/numpy.

[7] SciKit-Learn Framework. https://scikit-learn.org/.

[8] Linear Regression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html.

[9] Logistic Regression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[10] M. Bishop. Logistic Regression, Pattern Recoginition and Machine Learning.

[11] ELM Structure. https://www.sciencedirect.com/science/article/pii/S0925231215011327.

[12] Convolutional Layers. https://keras.io/layers/convolutional/.

[13] Cifar-10 CNN Example. https://keras.io/examples/cifar10_cnn/.

[14] TSNE Embedding. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html.

[15] Confusion Matrix. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.