Hesam Damghanian

Dr. S.R. Kheradpisheh

Predicting churn rate using neural networks

10 November 2020

Abstract

Churn rate can be predicted using customers' information, such as their balance, salary, credit score, and so on. The aim of this is project is to predict whether a customer will leave the service or not using their personal data.

Keywords: churn rate, TensorFlow, Deep Learning, Classification

1. Preprocessing

Given dataset had no null nor corrupted values and consisted of 14 columns, 3 of which were irrelevant ('RowNumber', 'CustomerId', 'Surname'), 2 were categorical ('Gender', 'Geography'), 5 were numerical ('CreditScore', 'Balance', 'EstimatedSalary', 'Tenure', 'NumberOfProducts'), and the rest were indicator or bucket columns.
Two methods of preprocessing were conducted; using TensorFlow and sklearn preprocessing tools. In order to feed categorical data to the network, it needs to be coded to numbers (e.g 1, 0). There are multiple methods to do so, such as One-Hot encoding, hash encoding, BaseN encoding and etc.

1.1 sklearn Method. I assumed the result of different combinations of encoding may be different and a combination exists that would give the best result. So I reproduced five different DataFrames(fig 1-5) with different types of categorical embeddings (also normalized numerical columns).

```
from sklearn.preprocessing import LabelEncoder , OneHotEncoder

data1 = data.copy()
data1.Gender = LabelEncoder().fit_transform(data.Gender) # Gender label encoding
data1.Geography = LabelEncoder().fit_transform(data.Geography)

data1.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	0.538	0	0	42	0.2	0.000000	1	1	1	0.506735	1
1	0.516	2	0	41	0.1	0.334031	1	0	1	0.562709	0
2	0.304	0	0	42	0.8	0.636357	3	1	0	0.569654	1
3	0.698	0	0	39	0.1	0.000000	2	0	0	0.469120	0
4	1.000	2	0	43	0.2	0.500246	1	1	1	0.395400	0

Fig. 1

```
data2 = data.copy()
data2 = pd.concat((pd.DataFrame(OneHotEncoder(sparse = False).fit_transform(pd.DataFrame(data.Geography)),columns = ["France","Spain","Germany"]),data2),axis = 1)
data2 = pd.concat((pd.DataFrame(OneHotEncoder(sparse = False).fit_transform(pd.DataFrame(data.Gender)),columns = ["Male","Female"]),data2),axis = 1)
data2 = pd.concat((pd.DataFrame(OneHotEncoder(sparse = False).fit_transform(pd.DataFrame(data.HasCrCard)),columns = ["Card","NoCard"]),data2),axis = 1)
data2 = pd.concat((pd.DataFrame(OneHotEncoder(sparse = False).fit_transform(pd.DataFrame(data.IsActiveMember)),columns = ["Active","Passive"]),data2),axis = 1)
data2.drop(columns= ["Geography", "Gender", "HasCrCard", "IsActiveMember"],inplace = True)
data2.head()
```

	Active	Passive	Card	NoCard	Male	Female	France	Spain	Germany	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	Exited
0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.538	42	0.2	0.000000	1	0.506735	1
1	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.516	41	0.1	0.334031	1	0.562709	0
2	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.304	42	0.8	0.636357	3	0.569654	1
3	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.698	39	0.1	0.000000	2	0.469120	0
4	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.000	43	0.2	0.500246	1	0.395400	0

Fig. 2

```
data3 = data.copy()
data3 = pd.concat((pd.DataFrame(OneHotEncoder(sparse = False).fit_transform(pd.DataFrame(data.Geography)),columns = ["France","Spain","Germany"]),data3),axis = 1)
data3 = pd.concat((pd.DataFrame(OneHotEncoder(sparse = False).fit_transform(pd.DataFrame(data.Gender)),columns = ["Male","Female"]),data3),axis = 1)
data3.drop(columns= ["Geography", "Gender",],inplace = True)
data3.head()
```

	Male	Female	France	Spain	Germany	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1.0	0.0	1.0	0.0	0.0	0.538	42	0.2	0.000000	1	1	1	0.506735	1
1	1.0	0.0	0.0	0.0	1.0	0.516	41	0.1	0.334031	1	0	1	0.562709	0
2	1.0	0.0	1.0	0.0	0.0	0.304	42	0.8	0.636357	3	1	0	0.569654	1
3	1.0	0.0	1.0	0.0	0.0	0.698	39	0.1	0.000000	2	0	0	0.469120	0
4	1.0	0.0	0.0	0.0	1.0	1.000	43	0.2	0.500246	1	1	1	0.395400	0

Fig. 3

```
data4 = data.copy()
data4.Gender = LabelEncoder().fit_transform(data.Gender) # Gender label encoding
data4 = pd.concat((pd.DataFrame(OneHotEncoder(sparse = False).fit_transform(pd.DataFrame(data.Geography)),columns = ["France","Spain","Germany"]),data4),axis = 1)
data4.drop(columns= ["Geography"],inplace = True)

data4.head()
```

	France	Spain	Germany	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1.0	0.0	0.0	0.538	0	42	0.2	0.000000	1	1	1	0.506735	1
1	0.0	0.0	1.0	0.516	0	41	0.1	0.334031	1	0	1	0.562709	0
2	1.0	0.0	0.0	0.304	0	42	0.8	0.636357	3	1	0	0.569654	1
3	1.0	0.0	0.0	0.698	0	39	0.1	0.000000	2	0	0	0.469120	0
4	0.0	0.0	1.0	1.000	0	43	0.2	0.500246	1	1	1	0.395400	0

Fig. 4

```
data5 = pd.get_dummies(data)
data5.head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Geography_France	Geography_Germany	Geography_Spain	Gender_Female	Gender_Male
0	0.538	42	0.2	0.000000	1	1	1	0.506735	1	1	0	0	1	0
1	0.516	41	0.1	0.334031	1	0	1	0.562709	0	0	0	1	1	0
2	0.304	42	0.8	0.636357	3	1	0	0.569654	1	1	0	0	1	0
3	0.698	39	0.1	0.000000	2	0	0	0.469120	0	1	0	0	1	0
4	1.000	43	0.2	0.500246	1	1	1	0.395400	0	0	0	1	1	0

Fig. 5

1.2 TensorFlow Method. TensorFlow also provides rich tools for preprocessing but is a bit different than sklearn. it casts DataFrames to tf.data.Dataset that has better integration with TF layers. Here I converted **geography** and **gender** columns to *indicator_column,* **age** to bucktized_column, and numeric columns to *numeric_column*. After splitting the dataset to train, test, and validation sets (all in dataframe format), converted them to tf.data.Dataset.

2. Building the network and hyperparameter tuning

The model is defined, compiled, and fitted in a function named create_model. It consists of 9 layers wrapped in a sequential model. The first layer is basically the feature layer (section 1.2) that has been created using TF *feature_column* and *DenseFeatures*. And next five layers are dense (fully connected) layers that their units and kernel regulators are chosen in the hyperparameter tuning section and their activation function is ReLU. At last, there is a dropout layer in order to reduce generalization error (regularizations) which its ratio is also chosen in the tuning section and a dense layer with only one unit, indicating the target 0,1 value (Exited column) which uses softmax as the activation function.

The optimizer of choice is also selected in the tuning part between [‘sgd’, ‘adam’, ‘RMSProp’]. The loss function is binaryCrossEntropy since the prediction the model is going to make is between two values. Next, I defined two callbacks, one is for early stopping in order to stop fitting the model when the accuracy of the model on train data is increasing but it’s not the same for the validation data (regularization), and the other callback is for using TensorBoard functions and graphs. (had some issue on graphing the network and unfortunately, it turns out that this is an issue on TensorBoard). The *run* function is the function that has to run the model several times, each time with a different set of parameters generated in the nested for loop section.

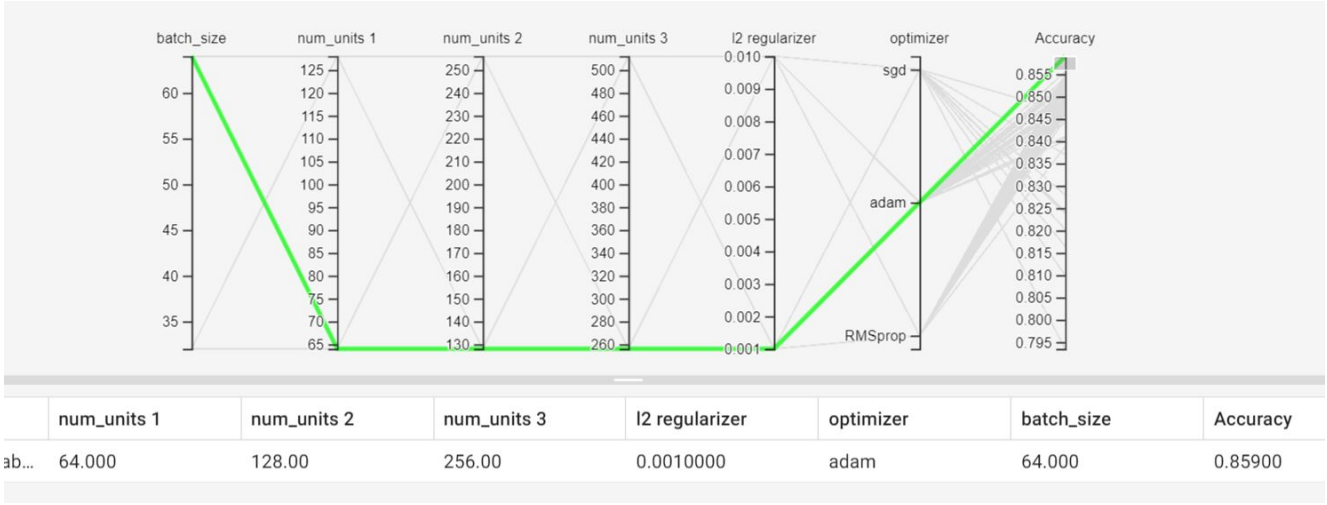
Table 1 shows the values that had been tested in the hyperparameter tuning.

	HP_NUM_UNITS_1	HP_NUM_UNITS_2	HP_NUM_UNITS_3	HP_BATCH_SIZE	HP_DROPOUT	HP_L2	HP_OPTIMIZER
Values	64, 128	128, 256	256, 512	32, 64	[0.2, 0.5]	[0.001, 0,01]	Adam, rmsProp, sgd
description	# Units per first and last dense layer	# Units per second and fourth dense layer	# Units per middle dense layer	Size of batches for training the model	Dropout rate range	L2	

Table 1

The result of this tuning is attached.

The figure below (Fig. 6) depicts the set of parameters that give the best results on the dataset created by tf.data.Dataset. And as we see in fig. 6 adam and rmsprop outperform sgd.



	num_units 1	num_units 2	num_units 3	l2 regularizer	optimizer	batch_size	Accuracy
ab...	64.000	128.00	256.00	0.0010000	adam	64.000	0.85900

Fig. 6 over 300 runs and the best result is shown in the figure

* Results of other settings are also available in TensorBoard part of the notebook.

And here is the result of five dataset variations (fig. 7). The Best accuracy on test data was 86%. (‘data0’) as shown in table 2.

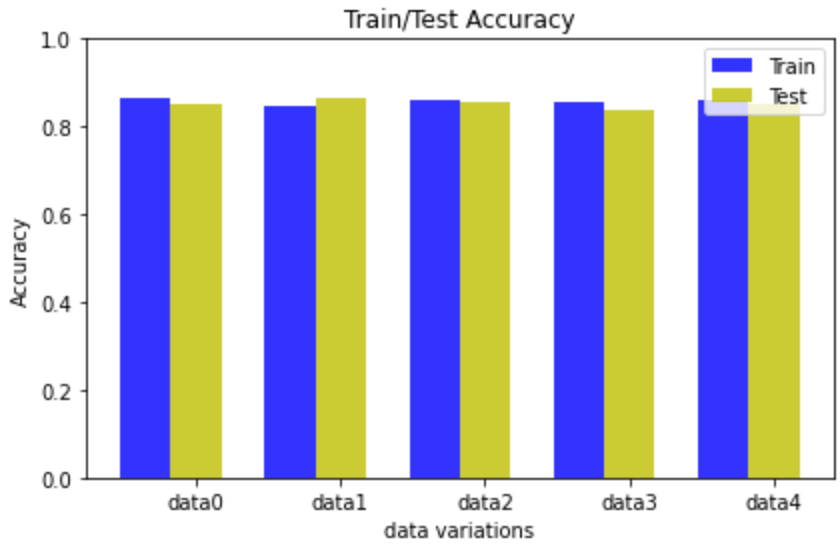


Fig. 7

	Data0	Data1	Data2	Data3	data4
Train accuracy	85.4	86	85.7	86.29	86.15
Test accuracy	84.2	85.6	86.20	83.20	84.6

Table 2

3. DISCUSSION

Churn rate can be predicted with relatively high confidence given the information of the customers.

The accuracy of 5 proposed dataset encoding was not significantly different, so the assumption that was made is wrong.

Early stopping prevents overfitting. Figure 8 shows accuracy results when the patience of the layer was 3 and figure 9 shows accuracy when the patience of the layer was 5. As we can see in fig. 9 the model is overfitted on the train data.

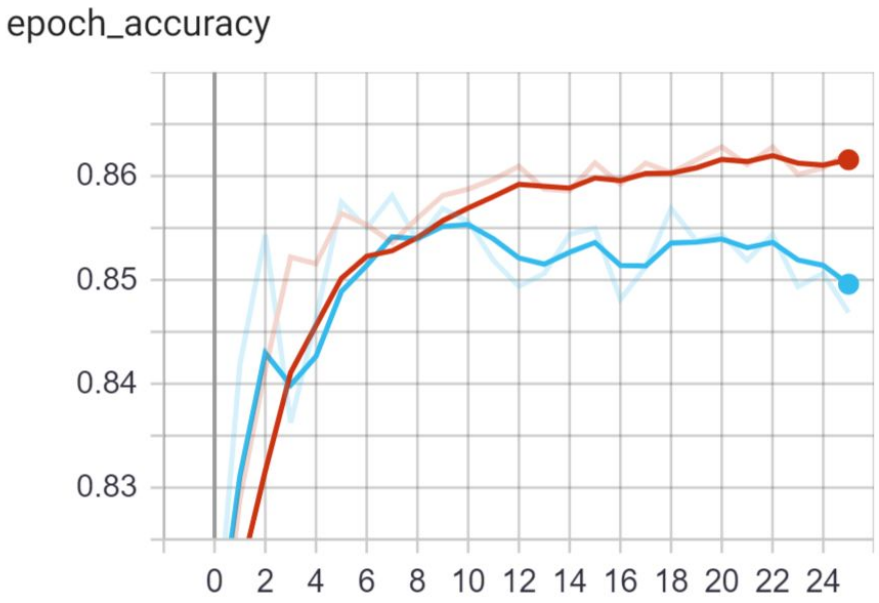


Fig. 8: the blue line is accuracy over validation and the red line is accuracy over train dataset

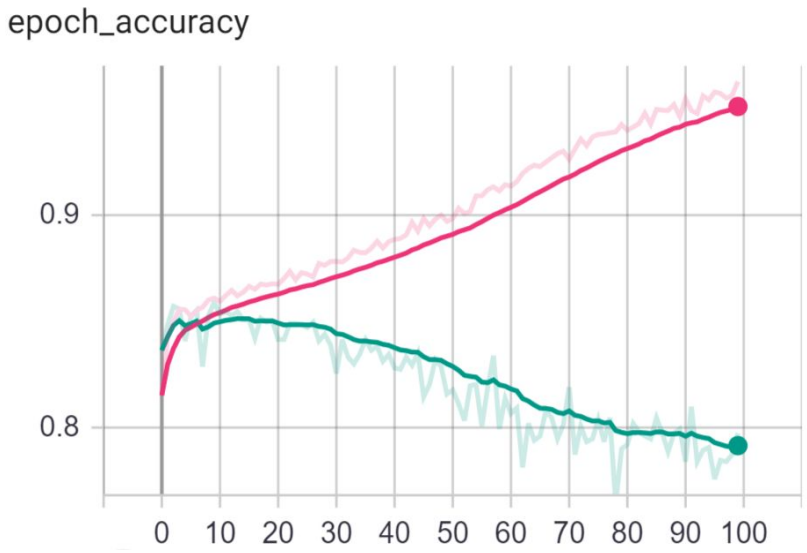


Fig. 9: the green line is accuracy over validation and the red line is accuracy over train dataset

It is worth mentioning that ensemble methods can easily outperform this expensive model, given the nature of the problem as table 3 shows.

	histGradient Boost	XGBoost	AdaBoost	GradientBo ost	This project. (best reults)
Train acc.	90.85	100	86.28	87.42	
Test acc.	89.8	83.4	84.65	86.05	86.20

Table 3.

* Ensemble methods were not tuned and methods like histGradientBoost gave 100 acc. on test dataset with n_estimator=1000.