HTML Final Project Report

members and work Loads:
B08202029 楊欣翰: data preprocessing, SVM model, writing report
B08202033 楊智凱 feature selection, random forest, writing report
B08901092 蔡東翰: data cleaning, DNN model, writing report
Coding language and package: Python. Package:libsvm[1], Pandas(pd)[2], sklearn[3], Numpy(np)[4]

Data Preprocessing

Combine all data set into an list:

We use `pd.merge()` to join each .csv file by using user ID.

Data cleaning:

Considering the missing data , by human learning, we've found that there are some features hinting at each other, which can help us fill in some of the missing data with relatively reasonable numbers. Belowings are how we handle the features being hunted. For the other features, we use "median" strategy to fill up the blanks.

Feature	**How to handle with the missing feature**
"Age"	If "Under 30" is "Yes", assign it with the average of 19 to 30.Else if "Senior Citizen" is "Yes", assign it with the average of 65 to 80.
"Longitude" "Latitude"	If "Lat Long" isn't "Nan", assign them with the corresponding value in "Lat Long". Else if the location has the same zip code as the other location, assign them with that location's longitude and latitude.
"Referre a friend"	If "Number of Referrals" is larger than 0, assign it with "Yes". Else if "Number of Referrals" is just 0, assign it with "No".
"Number of Referrals"	If "Referred a friend" is "Yes", assign it with the average value of positive "Number of Referrals". Else if "Referred a friend" is "No", assign it with 0.
"Tenure in Months"	If both "Total Long Distance Charges" and "Avg Monthly Long Distance Charges" are neither "Nan" nor 0, assign it with "Total Long Distance Charges" / "Avg Monthly Long Distance Charges". Else if both "Total Charges" and "Monthly Charge" are neither "Nan" nor 0, assign it with "Total Charges" / "Monthly Charge".
"Phone_Service"	If "Avg Monthly Long Distance Charges" is larger than 0 or "Multiple_Lines" is "Yes", assign it with "Yes".
"Internet_Service"	If "Internet_Type" isn't "Nan" or one of the other features clinging to Internet service isn't "No", assign it with "Yes"
Others	Following "Total Revenue" = "Total Charges" - "Total Refunds" +"Total Extra Data Charges" + "Total Long Distance Charges".

Handling features with strings

Some of the features are not numbers, they are strings instead. For example, the "Married"feature in "services.csv"file has "yes","no"possibilities. While most of the machine learning(ML) models are suitable for numbers instead of strings. Therefore, We try two ways to deal with this problem.

(1) One-hot encoding

If there are n possibilities in a feature, it would expands to n features. Take the "Married"feature for example, it becomes [1,0] if the answer is yes, [0,1] if the answer is no. Therefore, there are two features representing the original "Married"feature. We use `pd.get_dummies()` to do so.

(2) Hashing

Take the “Married” feature for example again. If the answer is yes, it will hash into 0, and the answer no would become 1. We use `pd.util.hash_array(users["Married"].to_numpy())` to do so.

After testing, we find out that one-hot encoding has higher f1-score and accuracy on validation set. We guess it is because most of the string data feature’s possibilities don’t have hierarchical relationships. Changing them into numbers might cause the models to pick by number size. Although one-hot encoding creates more features, models can separate different answers more easily. After one-hot encoding, the whole table contains training user ID becomes a 4226 rows(users) * 63 columns(features) table.

Data normalization

Since each feature in the dataset varies a lot. It is hard for ML models to find relationships between them. Therefore, we normalized every features.We use `StandardScaler()` from sklearn package to do so.

Data transformation

In class, we know that data transform might increase the performance of ML models. Therefore, we do two degrees transform on all dataset. We use `PolynomialFeatures(degree=2, interaction_only=False)` from sklearn to do so. After transformation, the whole table contains training user ID becomes a 4226 rows(users) * 1953 columns(features).

Feature selection

The number of features is large, and may cause a burden. To handle this “curse of dimensionality”, we tried to do feature selection to reduce the number of effective features that are really relevant to the label. Naively, we first eliminate the features whose meaning is not understandable for us. Second, we remove some redundant features, like the “age” and “Under 30”, where the information of the latter one is obviously contained in the former one.The next step we tried was to use some well-known feature selection algorithms to select features.

The first method we tried is eliminating low-variance features, which implies that this feature may be less relevant to the label. However, since the different feature values have different scales, which will be unfair when we do the elimination. Here comes the importance of data normalization, which paved a way for us to use the variance to do feature selection. Practically speaking, we used VarianceThreshold() method in the sklearn package to do this, and use the mean of the variance of the whole features as the threshold.

The last way we tried is to employ the Mutual Information (MI) score to quantify how relevant the features and the label of data are. MI score is defined as following:

$$MI(U,V) = \sum_{i=1}^{|U|}\sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{N} \log \frac{N|U_i \cap V_j|}{|U_i||V_j|}$$

One problem is that the mathematical definition of MI score is complicated. This problem can be resolved too, since there are existing package that can calculate the MI score. Specifically, we used mutual_info_regression in sklearn.feature_selection to do this job. The performance when we apply this selection method is good, so we adopted this method.

Handling imbalance data

We found that the labels in the dataset are imbalanced. There are about 75% of “No Churn”labels. Since the judging system is using f1-score instead of total accuracy; moreover, many ML models adjust via accuracy. Thus we decide to use an over-sampling method to increase the size of less-frequent labels. We use `imblearn.over_sampling` from the sklearn package to do so. Different kinds of oversampling methods are included in this package, we experiment all of it and find out that random oversampling method is better.

Training data and Validation data

For all 4226 training data, we use 10% of it to be a validation data set.

Model 1: Support Vector Machine (SVM)

We use the libsvm package to do SVM work. We tried three different kinds of SVM: linear, polynomial, gaussian SVM to train and predict the labels. We use “grid search ”which is mentioned in the libsvm paper to find the best coefficients for each parameter.

Linear SVM

libsvm parameter=`svm_parameter(f'-t 0 -c {C} -q')`
The gaussian SVM has the parameter C from soft margin SVM.
We tried different C and predicted on the validation set. Part of the result is in the tables below. The first row indicates the C coefficient.

0.0625	0.125	0.25	0.5	1	2	4	8	16
77.2511848341232	77.4881516587678	77.4881516587678	77.9620853080569	77.9620853080569	77.4881516587678	76.7772511848341	76.7772511848341	77.2511848341232

↑ Linear SVM with different coefficient’s accuracy

0.0625	0.125	0.25	0.5	1	2	4	8	16
0.284802786008805	0.287333533546329	0.28764987299828	0.29068635270035	0.305602706291972	0.290770271484865	0.276589451856702	0.268096034400382	0.28744178483046

↑ Linear SVM with different coefficient’s f1-score
We found that around C=0.125~1 might have the best result. We tried C=0.19 ,the public score is 0.358.

Gaussian SVM

The gaussian SVM ‘s kernel : $K(x, x') = |\gamma(\mathbf{x} - \mathbf{x}')|^2$
libsvm parameter=`svm_parameter(f'-t 2 -c {C} -g {g} -q')`
The gaussian SVM has the parameter γ and the C from soft margin SVM.
We did grid search on $C = 2^{-5} \sim 2^{5}$, $\gamma = 2^{-15} \sim 2^{-1}$, and also predicted on the validation set. Part of the result is in the tables below.

0.0	0.0625	0.125	0.25	0.5	1.0	2.0	4.0	8.0
3.0517578125e-05	72.74881516587678	72.74881516587678	72.74881516587678	72.74881516587678	72.74881516587678	76.77725118483413	79.14691943127961	81.27962085308057
6.103515625e-05	72.74881516587678	72.74881516587678	72.74881516587678	72.74881516587678	76.54028436018957	79.14691943127961	81.04265402843602	81.04265402843602
0.0001220703125	72.74881516587678	72.74881516587678	72.74881516587678	75.82938388625593	78.67298578199052	80.80568720379146	81.04265402843602	81.51658767772511
0.000244140625	72.74881516587678	72.74881516587678	73.22274881516587	77.48815165876776	79.85781990521326	81.27962085308057	81.51658767772511	80.80568720379146
0.00048828125	72.74881516587678	72.74881516587678	74.64454976303317	77.96208530805687	80.33175355450237	80.80568720379146	80.80568720379146	79.38388625592417
0.0009765625	72.74881516587678	72.74881516587678	73.45971563981043	77.72511848341233	79.14691943127961	79.85781990521326	78.90995260663507	78.90995260663507
0.001953125	72.74881516587678	72.74881516587678	72.74881516587678	75.11848341232228	76.77725118483413	76.77725118483413	76.77725118483413	77.01421800947867
0.00390625	72.74881516587678	72.74881516587678	72.74881516587678	74.17061611374407	74.40758293838863	74.88151658767772	74.88151658767772	75.11848341232228
0.0078125	72.74881516587678	72.74881516587678	73.45971563981043	74.17061611374407	74.40758293838863	74.40758293838863	74.40758293838863	74.40758293838863

↑ Gaussian SVM with different coefficient’s accuracy

0.0	0.0625	0.125	0.25	0.5	1.0	2.0	4.0	8.0
3.0517578125e-05	0.14037494284407864	0.14037494284407864	0.14037494284407864	0.14037494284407864	0.14037494284407864	0.21348637015781924	0.2380904719009013	0.25518925518
6.103515625e-05	0.14037494284407864	0.14037494284407864	0.14037494284407864	0.14037494284407864	0.21123112659698026	0.2380904719009013	0.25298652236062896	0.25304878048
0.0001220703125	0.14037494284407864	0.14037494284407864	0.14037494284407864	0.20143072057965675	0.23277746276830324	0.25197401299350325	0.2525761731781451	0.30271311579
0.000244140625	0.14037494284407864	0.14037494284407864	0.15284533172010487	0.22077294685990337	0.24319083694083696	0.2539894891703446	0.30341953950626604	0.29841472561
0.00048828125	0.14037494284407864	0.14037494284407864	0.18182182985553771	0.22499310206436404	0.2461769115442279	0.2513007720711648	0.28139064203306524	0.25938116419
0.0009765625	0.14037494284407864	0.14037494284407864	0.15858576236743507	0.22320839518390234	0.23775506465912002	0.2587552168390492	0.23976314172392602	0.23875181422
0.001953125	0.14037494284407864	0.14037494284407864	0.14037494284407864	0.19134563937934723	0.21460187752322582	0.2136890070248718	0.2141898798875543	0.21731219543
0.00390625	0.14037494284407864	0.14037494284407864	0.14037494284407864	0.17438769414575864	0.1782771852604255	0.1865837346174425	0.1865837346174425	0.19154778648
0.0078125	0.14037494284407864	0.14037494284407864	0.15858576236743507	0.17438769414575864	0.17862175016986734	0.17862175016986734	0.17918264760576474	0.17996068694

↑ Gaussian SVM with different coefficient’s f1-score

After grid search, we find out that $C = 4$, $\gamma = 2^{-12}$ might have the best result. The public score is 0.241.

Conclusion

We tried three different kinds of SVM, all of them can achieve very low E_{in}. The average is around 97%, the highest one is in gaussian SVM, it even achieves 99% train accuracy. Lower E_{in} can be achieved by giving higher C,d, γ. However, lower E_{in} doesn't mean better performance at validation set, it usually causes overfitting. In fact, the simplest, linear_svm, gets the best public score. SVM is efficient for doing one calculation, while we need to do grid search to find the best coefficients. Therefore, it's less efficient after all.

Model 2: Random Forest (RF)

Model description:

We use RandomForestClassifier in the sklearn.ensemble package. As for parameter setting, we tried to adjust the number of trees, and the depth of the trees, and used both the accuracy and F1 score as criteria for fine tuning the hyperparameters.

Besides the adjustment mentioned in the foregoing paragraph, we also do the model pruning as discussed in the class, however, the performance after pruning didn't change a lot. Here's our preliminary results on the validation set accuracy:

depth/# of trees	5	6	7	8	9	10
1000	0.7657	0.7785	0.8000	0.8012	0.8062	0.8095
2000	0.7700	0.7868	0.8008	0.8034	0.8083	0.8095
3000	0.7676	0.7544	0.7989	0.8043	0.8071	0.8100
4000	0.7709	0.7856	0.7983	0.8034	0.8083	0.8093

As we can see, the best validation accuracy is 0.8100.

Additional Effort and Results:

In addition to the hyperparameter finetuning, we've done efforts to make the random forest model perform better. One of our efforts is to use oversampling to combat the negative effects of biased datasets. On the other hand, we found that the model which has better validation accuracy, may not have better public leaderboard performance as well. The reason is because the metric of the leaderboard is F1 score. Therefore, we started to use F1 score as one of the criteria evaluating our models. Our best result is using 10000 trees, with depth 10. This setting of the model gets accuracy 0.8760, and F1 score 0.3463 in our validation set. The following is the confusion matrix:

Predicted\True label	0	1	2	3	4	5
0	255	21	11	13	1	6
1	5	22	8	14	1	3
2	1	6	4	5	0	4
3	2	3	3	8	0	3
4	3	1	1	5	1	2
5	0	3	1	2	0	4

However, we found that on the public leaderboard, the performance got quite better. The public score was 0.30368 (and the private score was 0.33482), compared with the same setting but without oversampling, whose public score was only 0.25416, and private score 0.26170.

Conclusion:

Compared with SVM, we found that the random forest model has worse performance. However, as for fine-tuning the hyperparameter, the random forest model is much easier, and can still have acceptable accuracy. With random oversampling, the model can perform much better, which meets our expectation that oversampling can combat the problem of biased dataset. We also find an interesting result that RF is less constrained by the data size. We accidently train RF with the validation set, and the result is close to training with the training set.

Model 3: DNN

Using deep neural networks may help us find out some subtle hidden patterns between the features.

Loss function choosing:

Instead of using the common cross-entropy error, considering the scoring criteria, we then use **f1 score** as the loss function to process our DNN. (loss = 1 - f1 score). That is, we do gradient descent with respect to f1 score.

Combat overfitting: Use dropout(30%) + L2 penalty

Model size choosing:

Since the training data is not huge, we prefer a shallow model with few layers. After some trying, we've found that only 1 hidden layer will have much better performance on evaluation set. The followings are the model sizes that we've tried: (input layer size: 1953, output layer size: 6)

hidden layer	model size	training accuracy	training f1 score	evaluation accuracy	evaluation f1 score	score on kaggle
0	1953 * 6	0.63144	0.44781	0.57109	0.28312	0.26706
1	1953 * 20 * 6	0.82202	0.46187	0.78436	0.31927	0.29649
1	1953 * 60 * 6	0.83701	0.51117	0.79621	0.31627	0.30482
1	1953 * 150 * 6	0.83833	0.51001	0.79858	0.34310	0.30368
1	1953 * 350 * 6	0.84437	0.52311	0.78436	0.29821	0.29097
1	1953 * 600 * 6	0.85016	0.54155	0.79858	0.33459	0.30549
1	1953 * 900 * 6	0.85016	0.55279	0.78910	0.31922	0.28680
2	1953*150*60*6	0.90799	0.75364	0.75829	0.27237	0.26885
2	1953*600*20*6	0.91009	0.76207	0.76778	0.29011	0.28644
2	1953*100*100*6	0.91667	0.77495	0.75592	0.27440	0.28413
2	1953*1000*20*6	0.91325	0.77793	0.75829	0.29085	0.28146
2	1953*1000*600*6	0.91614	0.78140	0.75118	0.26727	0.28440

As the results shown, DNN has the best performance when the number of hidden layers is 1. Besides, when we fix it at 1, we have stable outcomes no matter what the size of the middle layer is.

Model hyper parameters tuning:

At first, we have the default hyper parameters: learning rate = 0.001, batch size = 80, epoch = 500, weight_decay = 1e-5, momentum = 0.9. However, when we use random search on those parameters (learning rate = [0.01, 0.001, 0.0001]; batch size = [50, 80, 100, 150]; epoch = [300, 500, 1000]; weight_decay = [1e-3, 1e-5]; momentum = [0.5, 0.9, 0.95, 0.99]), we don't find significant improvement nor any rule on the result. (evaluation accuracy = 75% ~ 80%, evaluation f1 score = 0.28 ~ 0.32)

Conclusion:

Although it is hard for us to enhance our performance significantly by changing its structure and parameters, we can also say that the DNN model is a relatively stable model that maintains a not-bad accuracy to handle this problem. Besides, although there is an obvious gap between the training f1 score and evaluation f1 score, the difference between the evaluation f1 score is relatively small, which suggests that we may predict the outcome in the test set to an extent.

Comparison and Conclusion

Data preprocessing

data cleaning + one-hot encoding + normalization + 2-degree transformation + over-sampling

Model Comparison

model	efficiency	scalability	interpretability	accuracy
SVM	low	medium	medium	medium
RF	medium	medium	low	high
DNN	low	relatively low	low	not-bad and stable

Recommendation and its cons and pros

Our recommendation model: random forest
pros: accuracy, easy to combat overfitting, acceptable efficiency for this dataset, less constrained by data size
cons: lower interpretability, higher computation for big data set
Our recommendation preprocessing
pros: mentioned above
cons: increase size and features of dataset, which make it's harder to train ML models

Reference:

1. https://www.csie.ntu.edu.tw/~cjlin/libsvm/
2. https://pandas.pydata.org/
3. https://scikit-learn.org/stable/
4. https://numpy.org/