Report

March 7, 2019

1 Kaggle Competition Midterm Report

1.1 Samuel Kaeser

1.1.1 What I Tried

The goal of this competition was to maximize the ratio of true-positive predictions to total positive predictions, given a dataset of 16583 data instances defined by 25 features. I performed a variety of steps attempting to solve this problem, ranging from data exploration, data preprocessing, testing different model types, and model hyperparameter tuning.

Data Exploration I began developing a solution to this problem by importing the data and displaying statistics about each column that seen in Table 1. This provided useful infromation about the contents and distribution of each column. For example, we see that columns like f2 have a relatively small range of values (1 to 7) while columns like f1 and f7 have much wider ranges (-1 to over 300000!). This implies that scaling the data may be necessary to find the best model, since the magnitude of these values can force the weights of their respective columns to be very small or very large.

Following this, I displayed each column in a histogram like that seen in Figure 1. This allows one to better visualize the distribution, see what values columns are centered around, and get a sense of where outliers are located. This also exposes qualities not seen in the statistics table, like the presence of gaussian noise in the otherwise binary column f14. We can also see that several columns like f2 and f11 appear to contain a relatively small amount of unique values, indicating these columns may need to be one-hot-encoded to represent non-numeric categorical information.

Data Preprocessing I developed several modified datasets using the information above. The collection of datasets used different permutations of scaling methods, encoding thresholds, the presence of polynomial features, and a selection from the original and generated (if any) features. See Appendix A for an example of the code used in my data preprocessing and transformation pipeline.

Encoding the categorical data was the first step in preprocessing the original data. I arbitrarily decided that any column in the training data possessing fewer than 20 unique values would be labeled as categorical, while the rest would be numeric. I found this strategy did not work, as some of the encoded columns in the test data possessed a different set of unique values than those seen in the training data. In some of the datasets generated, I simply treated these problem columns as numeric. In others, I chose instead to merge the training and test data before performing this encoding to make sure that each value present in both sets would have a categorical column. Naturally this meant that any values present in the test data and not in the training data would be filled exclusively with zeros and be useless during training. I assumed this to be unseen categorical and not numeric in those datasets. In addition, a special encoding procedure was conducted on f14, where two additional columns were added to the dataset – one for the value zero and one for the value 1 – to eliminate the noise before removing the column.

Once encoding was completed, I added second-degree polynomial features to some of the datasets. This addition was indiscriminant, meaning that numeric columns could be combined with categorical columns to produce a new numeric column.

After polynomial features were added (if at all), I scaled the numeric data. I alternated between no scaling, standardization (subtracting the mean and dividing by the standard deviation), and modified standardization (subtracting the median and dividing by the interquartile range) which is more robust to a mean and standard deviation skewed by the presence of outliers. Scaling the data helps keep the magnitude of weights relatively consistent, which is critical in several types of models.

The final step involved selecting a subset of the features from some of the datasets for use in training. Usually anywhere from 15-25 features were selected, but in the case of polynomial this would increase to 60-100. These features were selected based on how correlated they were with the target column, with the most correlated columns being selected for training. Again, this was not performed on all sets of data generated.

These preprocessing steps left me with a grid of data containing different qualities due to the varying pre-processing steps applied. This exposed the impact made by each pre-processing decision and determined which steps were ultimately more useful when making predictions on new data.

Model Selection I trained only a few model types: scikit-learns RandomForestClassifier, XGBoosts XGBClassifier, and scikit-learns Multi-Layered Perceptron Classifier. I attempted to implement a DataSet class which would allow the use of fast.ai's more robust neural networks, but was ultimately unsuccessful. During each model's training, I used grid search and cross-validation to test different combinations of hyperparameters to lessen the likelihood that the model defaults were underfitting or overfitting the data. These models were then compared using the area under the receiver-operator curve (AUC). Randomness in model generation and training was negated through- out this process by using a random seed of 42 whenever applicable.

To get a sense of how each preprocessing combination performed, I trained the RandomForestClassifier on the datasets. This model performed reasonably well on most sets, receiving scores hovering around 0.80 to 0.82. Once the viability of a dataset was evaluated like this, I moved on to more aggressive grid search cross-validation over hyperparameters in the XGBClassifier. This model almost always outperformed the RandomForestClassifier, and gave me my top scores of 0.91 on the test data. I also performed grid search cross-validation over scikit-learn's MLPClassifier on some of the datasets, but this model was frought with drawbacks like lengthy training times and extremely poor performance, sometimes reaching areas smaller than 0.5.

See Appendix B for the code used to conduct model selection.

1.2 What Worked

At the conclusion of testing, the most successful model was an XGBClassifier with 200 estimators, each with a depth of 8 boosted trees, a learning rate of 0.15, an l_1 regularization coefficient of 1, and an l_2 regularization coefficent of 0. (all other hyperparameters maintained their default values). This model distinguished itself in training by successfully predicting three additional true positive instances (over the second most successful model),but this came at the expense of an additional 17 false positive predictions. While this seems like a modest increase, this was likely due to overfitting. Looking at the drop in placement between the public and private leaderboards, my model failed to generalize as well as other submissions, resulting in a drop of 20 placements.

This model was developed on a dataset where the training and test data had been merged to facilitate one-hot-encoding of columns with values present in the test data but not in the training data as described above. This data had its numeric components scaled via the modified standardization as described above, but no feature selection was done and polynomial features were not added.

1.3 What didn't work

Several elements of the above methodology failed to produce useful results. Most of these occurred in model evaluation and preprocessing, but a few mistakes may have also been made during data exploration.

The model evaluation and preprocessing stages contained several issues that hindered model performance. First, the choice of evaluating a few MLPClassifiers turned out to largely be a waste of computing resources and time since this model was often incapable of performing better than random guessing. Second, the addition of polynomial features failed to produce conclusive improvements in model performance. This could be because higher order relationships with the target simply do not exist, or that my preprocessing steps were flawed and created false relationships which were given more weight than patterns that did exist. Third, my aggressive tuning of hyperparameters made training appear more successful than it actually was, but this was usally becuase the model was overfit and could not accurately estimate the test data. Finally, I did not evaluate many model types. I chose to focus primarily on the boosted trees provided by XGBClassifier and I did not attempt to test other models like linear regression, support vector machines, or other neural networks.

The data exploration stage was something I failed to spend adequate time preparing. As an example, I did nothing to address the gaussian noise seen in f14 in my initial exploration, but after manually encoding this column, model performance increased dramatically by an additional 3%. What seems more significant however, is the massive number of false positives my models generate. At their very best, my models were producing nearly twice as many false positives as true positives. This fact (combined with the knowledge that my peers were able to generate better models) could imply that there were some artifacts remaining in the data that my model was sensitive to. Alternatively, I was introducing these artifacts myself through a flaw in my pipeline, or I failed to test a wide enough variety of models on my transformed data.

1.3.1 Additional Notes

While the steps above may sound like there was a lot of thought given to each of the steps before proceeding to the next, this was not the case. I frequently returned to the exploration and pre-processing steps when models failed to perform well. This is generally bad practice, as this leads to overfitting models to the training data and – as the discrepancy in my public vs. private rank imply – worse performance in general.

Removing highly correlated columns was another step I neglected to take in the preprocessing stage. The presence of highly correlated columns with each other can mean that any additional noise present in these columns can create a strong impact in the final prediction of a test instance.

I made several assumptions about the data after exploration, such as a small numbers of unique values meaning the column was categorical and deciding to combine the training and test data to get these categories. I could have made more careful assumptions which may have improved model performance.

1.4 Table 1.

```
In [3]: train_dataframe, test_dataframe = import_dataframes()
        train_dataframe.describe(include='all')
```

Out[3]:

	Y	f1	f2	f3	f4
count	16383.000000	16383.000000	16383.000000	16383.000000	16383.000000
mean	0.942135	43007.775865	1.044375	11.770938	118323.581456
std	0.233495	33611.182771	0.264806	353.187115	4518.059755
min	0.000000	-1.000000	1.000000	1.770000	23779.000000
25%	1.000000	20311.000000	1.000000	1.770000	118096.000000
50%	1.000000	35527.000000	1.000000	1.770000	118300.000000
75%	1.000000	74240.500000	1.000000	3.540000	118386.000000
max	1.000000	312152.000000	7.000000	43910.160000	286791.000000

	f5	f6	f7	f8	f9
count	16383.000000	16383.000000	16383.000000	16383.000000	16383.000000
mean	1.044436	0.050052	117089.674113	169730.178600	1.041812
std	0.265601	0.293892	10261.292970	69396.677853	0.258226
min	1.000000	0.000000	4292.000000	4673.000000	1.000000
25%	1.000000	0.000000	117961.000000	117906.000000	1.000000
50%	1.000000	0.000000	117961.000000	128130.000000	1.000000
75%	1.000000	0.000000	117961.000000	234498.500000	1.000000
max	9.000000	10.000000	311178.000000	311867.000000	11.000000

	...	f15	f16	f17	f18
count	...	16383.000000	16383.000000	16383.000000	16383.000000
mean	...	25894.316914	119045.099005	184622.040835	1.047305
std	...	36086.993946	18321.987129	100590.811845	0.306239
min	...	25.000000	4674.000000	3130.000000	1.000000
25%	...	4554.000000	118395.000000	118398.000000	1.000000
50%	...	13234.000000	118929.000000	119095.000000	1.000000
75%	...	38902.000000	120539.000000	290919.000000	1.000000
max	...	311696.000000	286792.000000	308574.000000	18.000000

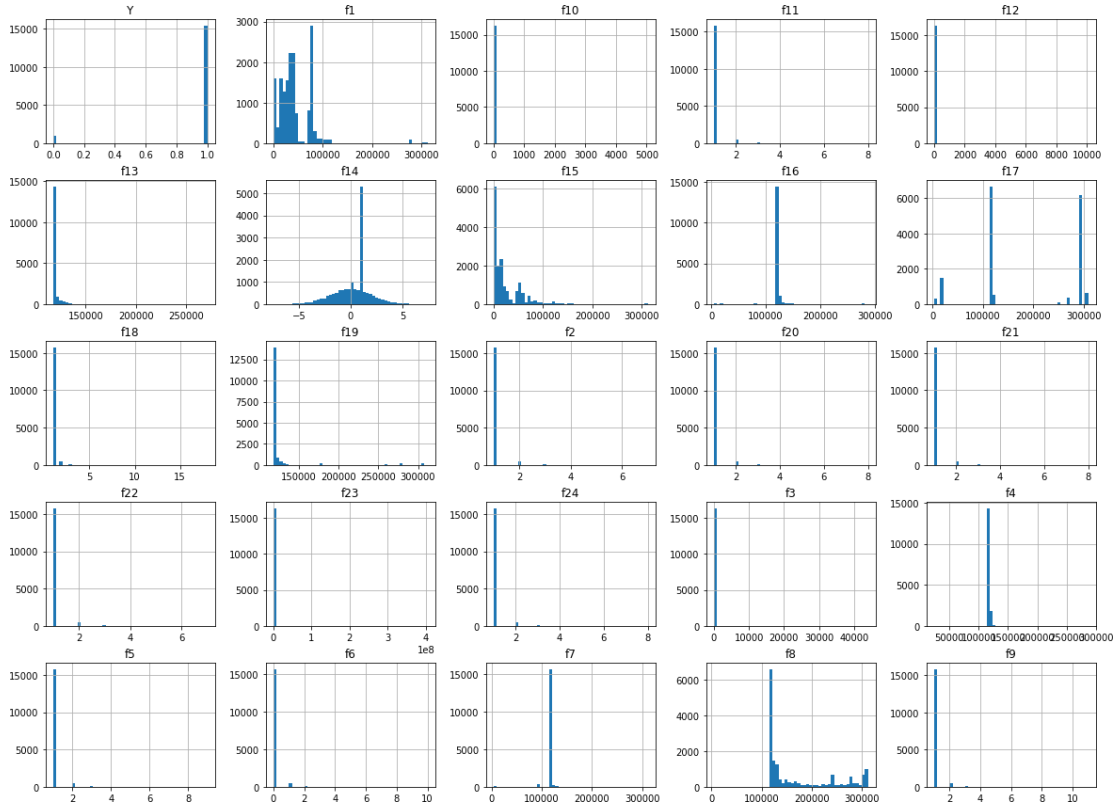
	f19	f20	f21	f22	f23	\
count	16383.000000	16383.000000	16383.000000	16383.000000	1.638300e+04	
mean	125959.667765	1.044558	1.045718	1.041934	3.271890e+04	
std	31091.344158	0.262576	0.266874	0.246597	3.184929e+06	
min	117879.000000	1.000000	1.000000	1.000000	1.000000e+00	
25%	118274.000000	1.000000	1.000000	1.000000	1.000000e+00	
50%	118568.000000	1.000000	1.000000	1.000000	2.000000e+00	
75%	120006.000000	1.000000	1.000000	1.000000	9.000000e+00	
max	311867.000000	8.000000	8.000000	7.000000	4.042886e+08	

	f24
count	16383.000000
mean	1.043948
std	0.259640
min	1.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	8.000000

[8 rows x 25 columns]

1.5 Figure 1.

In [6]:
```
display_column_distributions(train_dataframe)
plt.show()
```



1.6 Appendix A: Data Preprocessing pipeline

In [ ]:
```
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.preprocessing import  PolynomialFeatures


def  get_col_types(dataframe):
    # Find categorical columns
    cat_cols = []
    num_cols = []
    unique_cutoff =  20
    if  isinstance(dataframe,  pd.DataFrame):
        for column in dataframe.columns:
            if len(dataframe[column].unique()) < unique_cutoff and len(dataframe[colum
                cat_cols.append(column)
            else:
                num_cols.append(column)
    else:
        for  i  in  range(dataframe.shape[1]):
            if len(np.unique(dataframe[:, i])) < unique_cutoff and len(np.unique(dataf
                cat_cols.append(i)
            else:
```

```
                num_cols.append(i)

    return cat_cols, num_cols

def transform_data(X_train, y_train, X_test,
                   scaling=None,
                   poly_features=None,
                   feature_select=None):
    # Transform data based on scaling, poly feature, and feature selection instances
    train_idx = X_train.index
    test_idx = X_test.index

    all_idxs = train_idx.union(test_idx)
    all_df = get_merged_data(X_train, X_test)
    all_df.reindex(test_idx, axis='index')

    # Add categorical columns to data
    all_cat_cols, all_num_cols = get_col_types(all_df)
    all_df = pd.get_dummies(all_df, columns=all_cat_cols)
    X_train = all_df.drop(index=test_idx)
    X_test = all_df.drop(index=train_idx)

    # Transform input dataframes to numpy matrices
    if isinstance(X_train, pd.DataFrame):
        X_train = X_train.to_numpy()

    if isinstance(X_test, pd.DataFrame):
        X_test = X_test.to_numpy()

    # Transform data to include polynomial features
    if poly_features != None:
        print('Adding poly features...')
        X_train = poly_features.fit_transform(X_train, y_train)
        X_test = poly_features.transform(X_test)
    new_train_cat_cols, new_train_num_cols = get_col_types(X_train)

    # Scale data according to scaler
    if scaling != None:
        print('Scaling data...')
        X_train = scaling.fit_transform(X_train, y_train)
        X_test = scaling.transform(X_test)
        X_train_num = X_train.copy()
        for i in new_train_cat_cols:
            np.delete(X_train_num, i, 1)
        X_train_num_scaled = scalar.fit_transform(X_train_num)
        for main_idx, num_idx in zip(new_train_num_cols, range(len(new_train_num_cols)
            X_train[:, main_idx] = X_train_num_scaled[:, num_idx]
        X_test_num = X_test.copy()
```

```
            for i in new_train_cat_cols:
                np.delete(X_test_num, i, 1)
            X_test_num_scaled = scalar.transform(X_test_num)
            for main_idx, num_idx in zip(new_train_num_cols, range(len(new_train_num_cols)
                X_test[:, main_idx] = X_test_num_scaled[:, num_idx]

        # Select the desired number of features to leave in the dataset
        if feature_select != None:
            print('Selecting features...')
            X_train = feature_select.fit_transform(X_train, y_train)
            X_test = feature_select.transform(X_test)

        print('Done.')
        return X_train, X_test

    scalar = RobustScaler()
    X_train_all_scaled, X_test_all_scaled = transform_data(raw_X_train_strip,
                                                           y_train,
                                                           raw_X_test_strip,
                                                           scaling=scalar)
```

1.7 Appendix B: Model Evaluation

In [ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, roc_curve
from sklearn.model_selection import cross_val_predict

y_train_list = y_train.to_numpy().reshape(y_train.shape[0])

def run_grid_search(X_train, y_train, model, params, display_params=True, display_roc=
    grid = GridSearchCV(model, param_grid=params, scoring='roc_auc', n_jobs=-1, verbos
    grid.fit(X_train, y_train)
    if display_params:
        print(grid.best_params_)
        print(grid.best_score_)
    if display_roc:
        y_train_pred = cross_val_predict(grid.best_estimator_, X_train, y_train_list,
        print(confusion_matrix(y_train_list, y_train_pred))
        fpr, tpr, thresholds = roc_curve(y_train_list, grid.best_estimator_.predict_pr
        plot_roc_curve(fpr, tpr)
    return grid

def output_result(estimator, X_test, fname='out.csv'):
    y_pred = estimator.predict_proba(X_test)[:, 1]

    y_df = pd.DataFrame(data=y_pred, index=raw_X_test.index, columns=['Y'])
    y_df.to_csv(fname,columns=['Y'], index_label='Id', header=['Y'])
```

```
        def plot_roc_curve(fpr, tpr, label=None):
            plt.plot(fpr, tpr, linewidth=2, label=label)
            plt.plot([0, 1], [0, 1], 'k--')
            plt.axis([0, 1, 0, 1])
            plt.xlabel('False Positive Rate')
            plt.ylabel('True Positive Rate')
```

In [ ]:
```
random_seed = 42
xgb_params = {
    'max_depth': [4, 6, 8],
    'learning_rate': [0.1, 0.15, 0.2],
    'n_estimators': [300, 400, 500, 600, 700],
    'n_jobs': [-1],
    'random_state': [random_seed]
}

# Run grid search on XGBClassifier using scaled and selected data
xgb_grid_all_scaled = run_grid_search(X_train_all_scaled, y_train_list, XGBClassifier(
plt.show()
```

1.8 Appendix C: Report Setup Code (Run First!)

In [1]:
```
%%html
<style>
table {float:left}
</style>
```

```
<IPython.core.display.HTML object>
```

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def import_dataframes():
    train_dataframe = pd.read_csv('train_final.csv', index_col=0)
    test_dataframe = pd.read_csv('test_final.csv', index_col=0)
    return train_dataframe, test_dataframe

def display_column_distributions(data):
    data.hist(bins=50, figsize=(20,15))
```