CSE447 Final Project Report

Keren Luo, Xinyu Gan
kel417@lehigh.edu, xig417@lehigh.edu

1. Data Preparation

Features: ['id', 'name', 'lvl1', 'lvl2', 'lvl3', 'description', 'price', 'type']

- Preparing/cleaning data
  1. Text feature cleaning

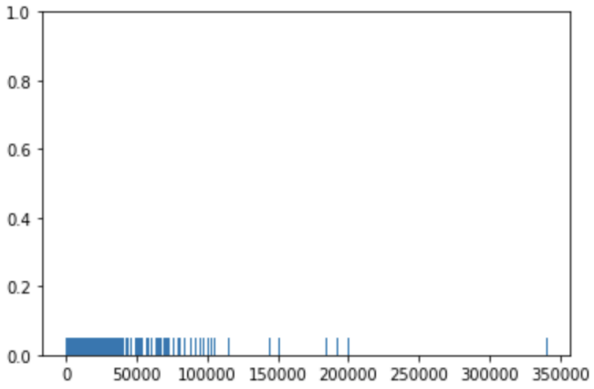
There might be some html tags such as been saved in the description feature since it directly been captured from webpages. Before we use this feature, we have to delete those html tags. Another situation is there are only '/' to split each record and no html tags in description. So first we use if statement to separately consider those conditions. Then delete html tags or just split splash and save record in a list and return.

  2. Numerical feature cleaning

We noticed that there are some abnormal values in price feature. So, we decided to set upper and down limit to filter out those too high or too low values in price feature.

- Percentile:

Find the [0,25,50,75,100]th percentile of the price feature. And get the IQR: Interquartile Range for later use in scale. The IQR is the range between the 1st quartile (25th percentile) and the 3rd quartile (75th percentile). The upper limit is set to be the 3rd quartile plus 1.5 times IQR and the lower limit to be -1 (all negative value in the dataset is -1). All the outliers is treated as missing value and processed later.



Price distribution

- Feature engineering
  1. Numerical features: ['price']
     - SimpleImputer:

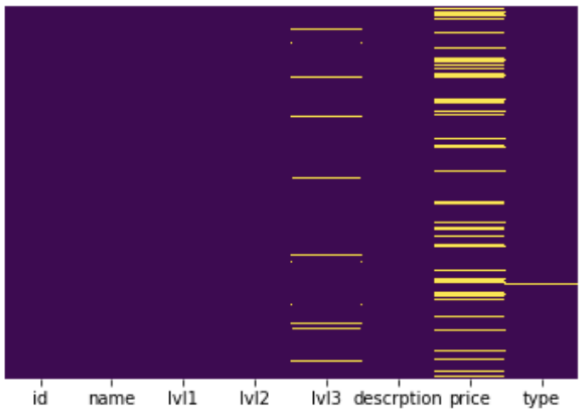
Find all missing values (including outliers we found before) in price feature and replace them using the median since it is numerical value.

     - RobustScaler:

Scale features using statistics that are robust to outliers. Center to the median and component wise scale according to the interquartile range. After scale, all price features are on the same scale relative to one other.

  2. Categorical features: ['lvl1','lvl2','lvl3','type']
     - SimpleImputer:

Find all missing values in those categorial features and replace them using string "missing".



Get null value from train

- OneHotEncoder:

  Encode categorical integer features as a one-hot numeric array. We need to transfer those categorical features from string to a sparse matrix composed by 0 and 1. And we set the "handle_unknown" parameter to be "ignore", if encounter some unknow categorical feature during the transform, the resulting one-hot encoded columns for this feature will be all zeros.

3. Text features: ['description'] ['name']

For text features, we need to apply algorithm in nature language processing. First, tokenize the text and get the count of occurrence of each word in the whole corpus. Then base on the matrix of token counts to calculate the frequency of those tokens which been called TF: term frequency.

- CountVectorizer:

  Convert a collection of text documents to a matrix of token counts.

- TfidfTransformer:

  Transform a count matrix to a normalized tf or tf-idf representation. Another refinement on top of tf is to downscale weights for words that occur in many documents in the corpus and are therefore less informative than those that occur only in a smaller portion of the corpus. This downscaling is called tf–idf for "Term Frequency times Inverse Document Frequency".

```
#%% Count Vectorizer

count_vect = CountVectorizer()

X_train_counts = count_vect.fit_transform(desc_train)
print(count_vect.get_feature_names())

#%% TFIDF

tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

Original code

```
get_desc_data = FunctionTransformer(lambda x: x['descrption'], validate=False)
desc_features = ['descrption']
desc_transformer = Pipeline([
            ('selector', get_desc_data),
            ('vect', CountVectorizer()),
            ('tfidf', TfidfTransformer()),
          ])
```

Integrated with Pipeline

2. Models

After we built one model, we use cross validation to get a score. It would return an array of scores of the estimator for each run of the cross validation. When we set cv = 5, we get the mean of those five scores which can directly show the accuracy of that model.

This function worked well when we use single model, the LogisticRegression Model have the highest mean_score of cross validation and the predict submission corresponding have best entry on Kaggle. But when we try to use double layer classifiers, it failed. The mean_score is pretty high but the result on Kaggle is bad.

```
#%% Fitting & Cross Validation
clf.fit(train,target['score'])

print(np.mean(cross_val_score(clf, train, target['score'], cv=5)))
```

Fitting & Cross Validation

- Successful models/algorithm
  - LogisticRegression:

    We defined a pipeline to combine numerical, categorical, description and name feature together.

```
preprocessor = ColumnTransformer(transformers=[('num',numeric_transformer,numeric_features),
                                               ('cat',categorical_transformer,categorical_features),
                                               ('desc',desc_transformer,desc_features),
                                               ('name',name_transformer,name_features),
                                               ])

clf = Pipeline(steps=[('preprocessor',preprocessor),
                      ('classifier',LogisticRegression(solver='lbfgs')),
                      ])
```

Pipeline classifier

- Failed models/algorithm
  - Single layer models:

    Naïve Bayes: This model is more suitable for discreate features and works very well when dealing with text and categorical features independently. However, its accuracy drops fast when the continuous numerical feature is added.

    SVM: This model works fine, but it takes a lot of time to train the data, and the results is not as good as Logistic Regression.

    Decision Tree: The result is ok, but still not be best model.

    K-Neighbors: The complexity of the dataset may lower the results of this model.

    Ensemble methods: Ada Boost works best because it fits additional copies of the classifier on the same datasets and focus more on different cases. We also tried Gradient Boosting, Random Forests and Extra Trees but none of them gets better result than Ada Boost.

  - Double layer models:

    1st layer: [Logistic Regression, Ada Boost, Random Forest, Extra Trees, Gradient Boosting]

    2nd layer: XGB Classifier

```
#%% Second layer
train2 = np.concatenate((predict_ab_train[:,1].reshape(-1,1), predict_et_train[:,1].reshape(-1,1),
                         predict_gb_train[:,1].reshape(-1,1), predict_lr_train[:,1].reshape(-1,1),
                         predict_rf_train[:,1].reshape(-1,1)), axis = 1)
test2 = np.concatenate((predict_ab_test[:,1].reshape(-1,1), predict_et_test[:,1].reshape(-1,1),
                         predict_gb_test[:,1].reshape(-1,1), predict_lr_test[:,1].reshape(-1,1),
                         predict_rf_test[:,1].reshape(-1,1)), axis = 1)
train_mean = np.mean(train2, axis = 1)
test_mean = np.mean(test2, axis = 1)
train22 = pd.DataFrame(train2)
test22 = pd.DataFrame(test2)

clf = xgb.XGBClassifier().fit(train2, target['score'])
predict = clf.predict_proba(test2)
```

Second layer classifier

When we consider the double layers model, we want to combine all predictions from five good worked single model and see if we can get better result. We tried to build a double layer model. We choose to combine predictions from 5 models in first layer as the input of the second layer. Then we use xgboost in second layer. We suppose this would get better result. It actually gives us the highest cross validation score. But it's ranked very bad in Kaggle public leaderboard around 3.7. Since the score of cross validation shows the relation between the target and the predict result from the built model, this double layer model may get overfitting for train data.

3. Results

- Cross validation score

LogisticRegression	0.7939480624686073
LinearSVC	0.7523737280991277
DecisionTreeClassifier	0.7091091754018759
KNeighborsClassifier	0.7288123177989296
AdaBoostClassifier	0.7927350907989122
GradientBoostingClassifier	0.7657794560655782
RandomForestClassifier	0.6948902430087563
ExtraTreesClassifier	0.6944492438915484
1st layer: [LogisticRegression, AdaBoostClassifier, RandomForestClassifier, ExtraTreesClassifier, GradientBoostingClassifier] 2nd layer: XGBClassifier	0.9437737392948117

- Final score on Kaggel

	Public Leaderboard	Private Leaderboard
Score	0.43621	0.45045

- Interpretation of the results
  The reason why logistic regression wins among all the model is its ability to handle both dense and sparse input. Since our text and categorical features are CSR sparse matrix and numerical feature are dense array. This model correlated the loosely related features and gets optimal performance.

4. Lesson learned

It is the first time we try to handle data project using python. The whole csv file contains different type of data included numerical, categorical and text.

When we started the project, we tried to clean all features like combine all 3 levels to be one feature and transfer level and type feature from string to integers. But latterly we realized some of those are actually unnecessary. So we decide only to clean the html tags in description feature and replace the abnormal values in price with its mean. During cleaning the description feature, it applied the functions used in previous homework to find certain punctuations like

We have tried different models to look for best result. In single model, the LogisticRegression have the best predictions. And the double layer classifier is new to us, we haven't tried that kind of model before. We thought it would improve the model since it combines 5 model and get second classifier. But it failed. This double layer model got very high predict accuracy on train data but bad on test result. We think it could cause the overfitting.

This project is a very good model to learn models from sklearn. We can easy use different models since the APIs have already been set. And the parameters are ready for us to use too. But with all these models, it is hard to choose which one would be good without trying. As new learners, we have to try all models to compare and filter out those models performed good. It may be better if we try to handle more data projects like this.

After we finish first complete raw code, we want to improve it. So we use different data cleaning methods and different models. It is all processing for better result. And we also improve our code with integrate like use Pipeline to combine functions use in one feature and features use in one classifier. And when we test with cross validation, we also try k-folds. This is a very thoughtful experience for us to try data science. We learn many useful functions related to data and those classifier models.