Predicting URL Relevance

STATS 202: Data Mining & Analysis, Final Project

Boyao Wang, Janmay Panchal

Team: Beryex

August 9, 2024

1 Introduction

Search engines have become an integral part of our daily lives, offering relevant documents and websites based on user input and browsing behavior, such as cookies. To determine the relevance of a document, search engines utilize hundreds of signals, ultimately returning a ranked list of documents based on these signals.

In this project, which is a Kaggle competition[1], we are provided with a training dataset comprising 80,046 observations and 10 attributes, including “query_length”, “is_homepaged”, and eight unnamed signals. The dataset also includes a binary output indicating whether an observation is relevant, based on search engine query and URL data. Additionally, we have a test dataset containing 30,001 observations with the same 10 attributes, but without the relevance output.

Our objective is to develop a model using the training dataset to predict the relevance of each observation in the test dataset. Our codes are available on GitHub[1].

2 Dataset Preprocessing

Before modeling the training dataset, we first need to preprocess the data, including removing outliers, standardizing the data, and analyzing the attributes to be used. These steps help to ensure a consistent data scale and prevent the model from being affected by outliers or unnecessary attributes.

[1] Codes available at `https://github.com/Beryex/URL-Relevance-Prediction`

2.1 Standardize dataset

When we loaded the dataset 6.3, our first step was to standardize the data. This process ensures that different input attributes have the same scale, which is crucial for the model fitting process and facilitates subsequent processing 6.3.

2.2 Remove outliers

We initially used the interquartile range (IQR) method to remove outliers 6.3, defining any sample with one or more attributes falling outside the upper or lower quartile range as an outlier. However, we found that this approach resulted in the removal of 14015 samples (approximately 17% of total samples). Consequently, we relaxed the criteria, redefining outliers as samples with more than half of their attributes outside the quartile range, that is, 23 outliers in total. Despite this adjustment, the evaluation results showed that removing these outliers actually decreased our average cross-validation score.

2.3 Select attributes

We generated pairwise correlation plots 6.3 as seen in figure 1 to analyze the relationships between different attributes.

From Figure 1 6.1, we observed that the correlation plots of sig5 and sig6 with other attributes are very similar. This observation led us to investigate potential collinearity between sig5 and sig6. To select the most appropriate features, we employed forward stepwise feature selection 6.3, utilizing a spline model based on logistic regression combined with 5-fold cross-validation.

The results showed that the feature combination yielding the highest average cross-validation accuracy did not include sig5. Therefore, we decided to remove the sig5 attribute from the dataset 6.3.

Next, we aimed to enhance our model by adding additional attributes using the kernel method, specifically by including polynomial terms with degree=2 6.3. However, as shown in the Evaluation section, adding these extra attributes actually decreased our average cross-validation accuracy.

We then applied Principal Component Analysis (PCA) 6.3 to analyze and retain the attributes that best explain the variability (i.e., variance) in the original data. Figure 2 6.1 illustrates the relationship between the number of features and the explained variance. From the figure, we can see that all 9 linearly transformed attributes are needed to explain 99%

of the variance in the original data. However, as indicated in the Evaluation section, using PCA did not improve our average cross-validation accuracy.

3 Methods

To model the true relationship between attributes and URL relevance, we employed various methods and built multiple neural networks for learning. We also conducted extensive hyperparameter searches and used cross-validation to evaluate the performance of different methods. Additionally, we used a separate validation set, split from the training set, to assess the performance of the different neural networks. We set all random state to 1 to ensure reproducibility of results across different experiments.

3.1 Logistic regression

The first method we attempted was logistic regression 6.3, as it is a simple and useful approach for classification tasks. However, after conducting hyperparameter searches for penalty, C, solver, and max_iter, using 5-fold cross-validation average accuracy as criteria, the optimal hyperparameters are shown in Table 1 6.2. The corresponding average accuracy was only 65.37%, indicating that logistic regression is too simplistic to capture the complex relationship between attributes and URL relevance.

3.2 Random Forest

Next, we tried Random Forest 6.3, a slightly more complex model. We performed hyperparameter searches for n_estimators, max_depth, min_samples_split, and min_samples_leaf, and again used 5-fold cross-validation. The optimal hyperparameters found are listed in Table 2 6.2, and the corresponding average accuracy improved to 66.62%. This suggests that Random Forest can capture more complex relationships between attributes and relevance.

3.3 Boosting

We then tried boosting 6.3, and similarly performed hyperparameter searches for num_leaves, initial_learning_rate, n_estimators, and min_child_samples, again using 5-fold cross-validation. The optimal hyperparameters are listed in Table 3, and the corresponding average accuracy increased to 66.73%, showing a slight improvement. We also tried methods like the Voting

Classifier and the Stacking Classifier, but since their performance was inferior to boosting, they were not included in this report.

3.4 Neural Networks

Finally, we constructed a data loader 6.3 and built several neural networks 6.3, including VGG, and ResNet, with their convolutional layers replaced by linear layers and BatchNorm2D replaced by BatchNorm1D. Unlike previous methods, we did not use cross-validation to evaluate the models. Instead, we separated a validation dataset from the training dataset to assess model performance. We also conducted hyperparameter searches for the learning rate, optimizer's weight decay, and the number of training epochs. The resulting training loss curves are shown in Figure 3 6.1. As seen in the figure, the evaluate loss on validation set firstly decrease and the increase dynamically. The neural networks tended to overfit the dataset easily (with the highest validation accuracy reaching 68.34%) and exhibited training instability, even when we reduced the learning rate. To mitigate overfitting, we increased the optimizer's weight decay, adjusted the validation set ratio, and applied dropout layers. The final optimal hyperparameters are listed in Table 4 6.2. Although neural networks have strong representational power, they are prone to overfitting or failing to capture the true relationships. The final results were not as good as those achieved with the traditional methods mentioned above.

4 Evaluation

We evaluated the performance of all attempted methods under the optimal hyperparameters and assessed the impact of removing outliers, applying kernel methods, and using PCA on these methods. The results are shown in Table 5 6.2.

As observed, removing outliers (despite eliminating only 23 samples in total) resulted in performance declines across all models. We hypothesize that these outliers represent high-leverage extreme cases that are beneficial for the model's fitting. Additionally, using degree-2 kernel methods to add features also led to performance degradation, possibly due to the interference of redundant features with the model's fitting. Similarly, applying PCA did not improve model performance, which we attribute to the loss of original information when the attributes were linearly mapped into an orthogonal feature space.

Standardizing the data proved to be effective. While not standardizing the data might slightly improve the average cross-validation accuracy in some cases, standardizing clearly yielded better results on the test dataset. Removing sig5 was also beneficial; although it did not significantly affect the average cross-validation accuracy, it slightly improved the test accuracy.

5 Conclusion

In summary, for URL Relevance Prediction, traditional methods like Random Forest outperformed deep learning approaches, providing better and more stable results. Additionally, boosting achieved the best performance among all the methods tested.

6 Appendix

6.1 Figures

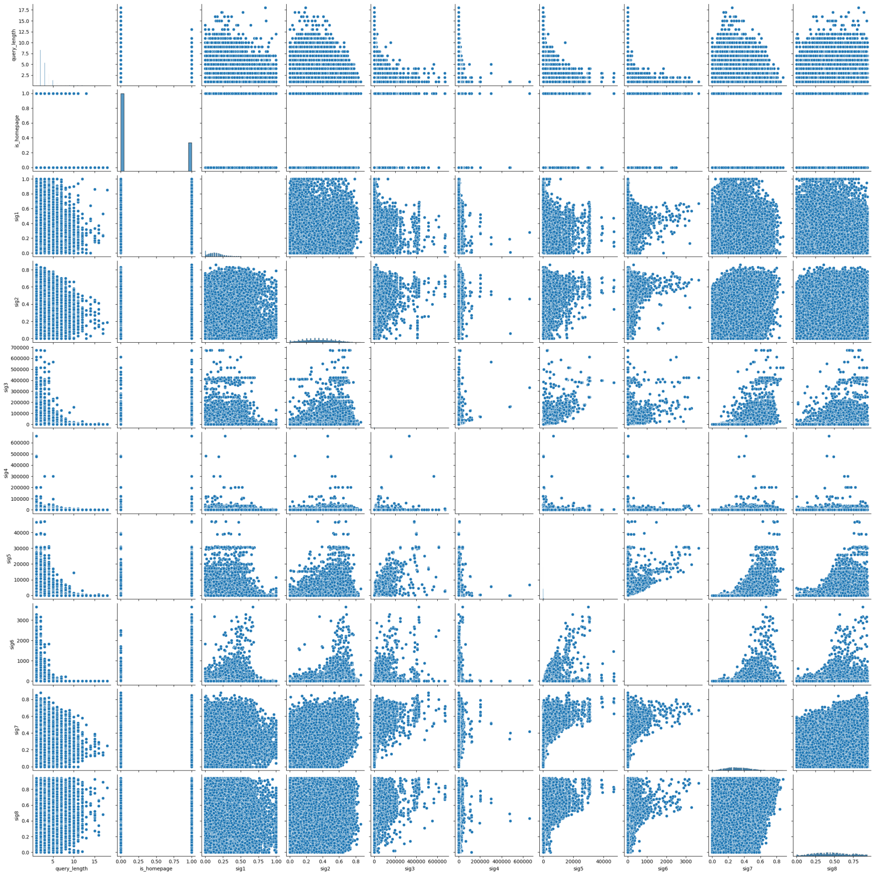

Figure 1: Pairwise correlation between attributes

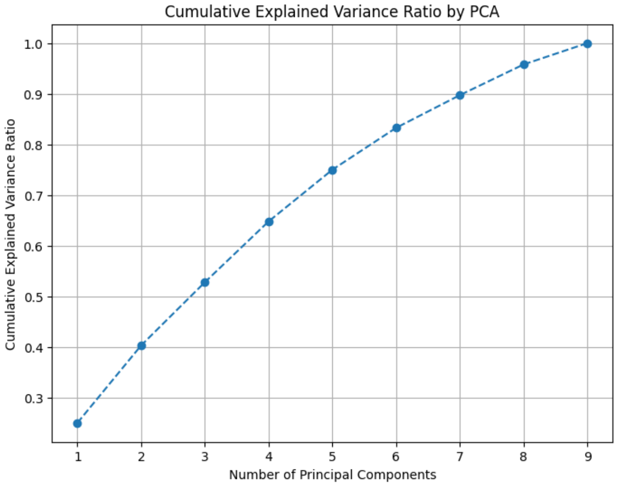

Figure 2: Pairwise correlation between attributes

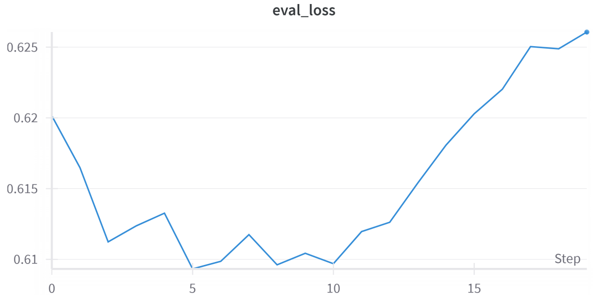

Figure 3: Evaluation loss on validation set for vgg19

6.2 Tables

Table 1: Optimal Hyperparameter for Logistic Regression

C	max_iter	penalty	solver	CV average acc
1	100	l2	lbfgs	65.37%

Table 2: Optimal Hyperparameter for Random Forest

n_estimators	max_depth	min_samples_split	min_samples_leaf	CV average acc
300	10	2	1	66.62%

Table 3: Optimal Hyperparameter for Boosting

num_leaves	learning_rate	n_estimators	min_child_samples	5 folder CV average acc
31	0.1	100	20	66.73%

Table 4: Optimal Hyperparameter for Neural Networks

model	initial_learning_rate	weight_decay	total_epoch	validation acc
VGG11	0.1	1e-4	10	66.66
VGG19	0.05	1e-2	20	66.72
ResNet18	0.05	1e-4	10	66.76
ResNet50	0.05	1e-2	20	66.23

Table 5: Dataset Preprocessing's Effect on Boosting with Optimal Hyperparameter

Standardize	Remove Sig5	Remove Outlier	Kernel Methods	Apply PCA	CV Average Accuracy	Test Accuracy
No	No	No	No	No	66.83	67.43
No	No	No	No	Yes	58.83	N/A
Yes	No	No	No	Yes	65.97	N/A
Yes	No	Yes	No	No	66.73	N/A
Yes	No	No	Yes	No	66.73	N/A
Yes	No	No	Yes	Yes	65.87	N/A
Yes	Yes	No	No	No	66.73	68.25
Yes	Yes	Yes	No	No	66.64	67.53
Yes	Yes	No	Yes	No	66.58	N/A

6.3 Codes implementation

To load the datasetset:

```
train_data = pd.read_csv(TRAIN_DATA_PATH)
test_data = pd.read_csv(TEST_DATA_PATH)

features = ['query_length', 'is_homepage', 'sig1', 'sig2',
   'sig3', 'sig4', 'sig5', 'sig6', 'sig7', 'sig8']
train_features = train_data[features]
train_labels = train_data['relevance']
test_features = test_data[features]
```

To standardize the datasetset:

```
scaler = StandardScaler()
train_features = scaler.fit_transform(train_features)
test_features = scaler.transform(test_features)
```

To remove the outliers from the dataset:

```
features_df = pd.DataFrame(train_features)
Q1 = features_df.quantile(0.25)
```

```
Q3 = features_df.quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

original_sample_num = train_features.shape[0]

outlier_mask = (features_df < lower_bound) | (features_df > upper_bound)
outlier_count = outlier_mask.sum(axis=1)
outlier_threshold = train_features.shape[1] // 2
mask = outlier_count <= outlier_threshold

train_features = train_features[mask.values]
train_labels = train_labels[mask.values]
washed_sample_num = train_features.shape[0]
print(f"We remove {original_sample_num - washed_sample_num}
      outliers from dataset")
print(f"Currently we have {washed_sample_num} total samples")
```

To visualize the relationships between the attributes:

```
sns.pairplot(train_data[features])
plt.show()
```

To run forward stepwise feature selection:

```
all_features = ['query_length', 'is_homepage', 'sig1', 'sig2',
              'sig3', 'sig4', 'sig5', 'sig6', 'sig7', 'sig8']
optimal_features = []
optimal_cur_features = []
optimal_pre_features = []
best_acc = 0
for idx in range(len(all_features)):
 best_cur_acc = 0
 for new_feature in all_features:
  cur_features = optimal_pre_features[:]
  if new_feature not in optimal_pre_features:
   cur_features.append(new_feature)
  else:
   continue
  random.seed(1)
  np.random.seed(1)
  train_data = pd.read_csv(TRAIN_DATA_PATH)

  cur_train_features = train_data[cur_features]
  cur_train_labels = train_data['relevance']

  spline_transformer = SplineTransformer(n_knots=100, degree=3)
  model = LogisticRegression(random_state=1)
  pipeline = make_pipeline(spline_transformer, model)

  kf = KFold(n_splits=10, shuffle=True, random_state=1)
  cv_scores = cross_val_score(pipeline, cur_train_features,
         cur_train_labels, cv=kf, scoring='accuracy')

  if best_cur_acc <= np.mean(cv_scores):
   optimal_cur_features = cur_features
   best_cur_acc = np.mean(cv_scores)
 optimal_pre_features = optimal_cur_features
 if best_acc <= best_cur_acc:
  optimal_features = optimal_cur_features
  best_acc = best_cur_acc
print(f"Optimal features combination: {optimal_features}
   with best acc: {best_acc}")
```

To remove sig5 from the dataset:

```
train_features = np.concatenate((train_features[:, 0:6],
         train_features[:, 7:]), axis=1)
test_features = np.concatenate((test_features[:, 0:6],
        test_features[:, 7:]), axis=1)
```

To add features using kernel methods:

```
poly = PolynomialFeatures(degree=2, include_bias=False)
```

```
train_features = poly.fit_transform(train_features)
test_features = poly.transform(test_features)
```

To apply PCA:

```
pca = PCA()
pca.fit(train_features)
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)

plt.figure(figsize=(8, 6))
plt.plot(range(1, len(cumulative_variance_ratio) + 1),
   cumulative_variance_ratio, marker='o', linestyle='--')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Cumulative Explained Variance Ratio by PCA')
plt.grid(True)
plt.show()

n_components = np.argmax(cumulative_variance_ratio >= 0.999) + 1
print(f"Number of principal components to retain
   (99% variance): {n_components}")

pca = PCA(n_components=n_components)
train_features = pca.fit_transform(train_features)
test_features = pca.transform(test_features)
```

To use logistic regression:

```
model = LogisticRegression(
 random_state=1,
)

param_grid = [
 {
  'penalty': ['l2'],
  'C': [0.01, 0.1, 1, 10, 100],
  'solver': ['newton-cg', 'lbfgs', 'sag'],
  'max_iter': [100, 300, 500]
 },
 {
  'penalty': ['l1'],
  'C': [0.01, 0.1, 1, 10, 100],
  'solver': ['liblinear'],
  'max_iter': [100, 300, 500]
 },
 {
  'penalty': ['elasticnet'],
  'C': [0.01, 0.1, 1, 10, 100],
  'solver': ['saga'],
  'l1_ratio': [0.2, 0.5, 0.8],  # Adding l1_ratio for elasticnet
  'max_iter': [100, 300, 500]
 },
 {
  'penalty': [None],
  'solver': ['newton-cg', 'lbfgs', 'sag', 'saga'],
  'max_iter': [100, 300, 500]
 }
]

grid_search = GridSearchCV(
 estimator=model,
 param_grid=param_grid,
 scoring='accuracy',
 cv=KFold(n_splits=5, shuffle=True, random_state=1),
 verbose=1,
 n_jobs=-1
)

grid_search.fit(train_features, train_labels)

print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validation accuracy: ", grid_search.best_score_)

best_model = grid_search.best_estimator_
best_model.fit(train_features, train_labels)
```

To use random forest:

```
model = RandomForestClassifier(
 random_state=1,
)

param_grid = {
 'n_estimators': [50, 100, 300],
 'max_depth': [5, 10, 20],
 'min_samples_split': [2, 4],
 'min_samples_leaf': [1, 2, 4],
 'max_features': ['sqrt'],
}

grid_search = GridSearchCV(
 estimator=model,
 param_grid=param_grid,
 scoring='accuracy',
 cv=KFold(n_splits=5, shuffle=True, random_state=1),
 verbose=1,
 n_jobs=-1
)

grid_search.fit(train_features, train_labels)

print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validation accuracy: ", grid_search.best_score_)

best_model = grid_search.best_estimator_
best_model.fit(train_features, train_labels)
```

To use boosting:

```
model = lgb.LGBMClassifier(
 random_state=1,
 verbosity=-1,
)

param_grid = {
 'num_leaves': [20, 31, 40],
 'max_depth': [-1, 5, 10],
 'learning_rate': [0.01, 0.1, 0.2],
 'n_estimators': [50, 100, 150],
 'min_child_samples': [15, 20, 25],
 'subsample': [0.8],
 'colsample_bytree': [0.8],
}

grid_search = GridSearchCV(
 estimator=model,
 param_grid=param_grid,
 scoring='accuracy',
 cv=KFold(n_splits=5, shuffle=True, random_state=1),
 verbose=1,
 n_jobs=-1
)

grid_search.fit(train_features, train_labels)

print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validation accuracy: ", grid_search.best_score_)

best_model = grid_search.best_estimator_
best_model.fit(train_features, train_labels)
```

To bulid dataloader for training neural network: See codes at `https://github.com/Beryex/URL-Relevance-Prediction/blob/main/final_project_neural_network.ipynb`

To train neural network: See codes at `https://github.com/Beryex/URL-Relevance-Prediction/blob/main/final_project_neural_network.ipynb`

References

[1] STATS 202. Stanford stats 202 prediction 2024. https://kaggle.com/competitions/stanford-stats-202-prediction-2024, 2024.