Application of Predictive Models in PE/VC Investments

FRE-GY 7043 Final Project Report

Donguk Shin ds6329@nyu.edu
Feng Su fs2426@nyu.edu
Guangyu Wang gw2138@nyu.edu
Hongzheng Li hl4607@nyu.edu
Kaiyun Kang kk3567@nyu.edu

Abstract

Private equity and venture capital (PE/VC) has always been a tricky topic for quantitative analysts. Data of startup companies from different vendors are usually inconsistent and difficult to combine. The nature of business data - large data size, broad spectrum, low signal-to-noise ratios - also adds to the difficulty of making effective investment decisions. Appropriate machine learning algorithms are needed to deal with these problems. Our project explores three application scenarios where machine learning models can prove useful in PE/VC research: approximate string matching, finding similar companies, and predicting capital raising (CapRaise) activities. We employ Natural Language Processing (NLP) models, such as TF-IDF, Word2Vec, Sentence-BERT, and LightGBM Ranker, for the first two tasks. For CapRaise prediction, we use a variety of classical and deep learning models, e.g., gradient-boosted trees and deep neural networks (DNN). We also test automated machine learning (AutoML) models to search on a broad range of models and hyperparameters. Appropriate performance measures (e.g. NDCG score, F1 score) are selected to compare the performances of models in each scenario. We find that ranker models and tree-based algorithms achieve the best overall outcomes in the matching part and the prediction part, respectively. Models with optimal performances are implemented in a web application where users can search for information of relevant companies through an interactive interface.

Keywords: Natural Language Processing (NLP), approximate string matching, ranking models, automated machine learning (AutoML)

1 Introduction and Objectives

This paper focuses on solving three problems of approximate string matching, finding similar companies, and predicting raising-money companies, based on using machine learning and natural language processing (NLP) techniques. Collaborating with other teams, we create a searching App for users who would like to search and see the most relevant companies' information by inputting keywords.

Firstly, approximate string matching is a technique that helps us identify two elements of strings that are approximately similar but are not exactly the same. In our research, we get our data from data vendors (e.g. BrightQuery), containing many small technical private company firmographic and financial information. However, in some datasets, there are various names for the same company and some missing values on the domain name column. Therefore, fuzzy matching algorithms are required to obtain a large dataset that left-merged all company information according to their company names.

In practice, our team apply Natural Language Processing (NLP) approaches with three different word embedders: N-gram & TF-IDF, Word2Vec, and Sentence-BERT. We also try ranking models by two methods: LightGBM Ranker and XGBoost Ranker. Comparing their performances with the baseline model of Databolt smart join, we evaluate all models by calculating NDCG scores for the top five matching items.

Secondly, to find similar companies for a given list of companies, we utilize our data including website text and content search information. For the purpose of increasing the possibility of matching company information correctly, we preprocess the content search column. Rather than applying NLP on company names that have only several words, this time we implement similarity analysis by TF-IDF vectorized embedding method on paragraphed long data.

Thirdly, we try to predict the future profitability of start-up companies with their history of fund raising activities. The information of past activities are captured by a time-series dataset including the topics, qualities, and web ratings. With these features, we test a variety of machine learning models, such as decision trees and neural networks, to predict whether these companies will be lucrative in the future.

2 The Fuzzy Matching Model

2.1 Background

In the process of data collection, we sometimes come across several different vendors providing data for a similar set of companies. Each vendor may have exclusive features about these firms, and it will be ideal if we are able to merge all these features into a comprehensive data set. By nature of using different vendors, however, there are usually no consistent identifiers to match corresponding companies.

There are either exact matching or fuzzy matching. Hall (1980) explained this classification as "when the name is known in exactly the form in which it is recorded in the directory, then looking it up is easy. But For similarity problems, different measures are surveyed, with a full description of the well-established dynamic programming method relating this to the approach using probabilities and likelihoods". In real-world data, even when the company names are essentially the same, there are usually tiny discrepancies such as acronyms or spelling differences, which prevents the use of exact matching methods. Therefore, we will have to merge these records with fuzzy matching techniques.

Some of the approximate string matching application areas are Image processing, word stemming, genetic algorithms, knowledge discovery, information retrieval, and pattern recognition (Kalyanathaya, 2019). As humans, we can easily spot typing errors or conceptually understand when two similar things are the same. Fuzzy matching algorithms try to help a computer do this. Rather than a match between two strings being exactly the same or not the same, fuzzy matching gives a similarity score. Baeza-Yates (1996) has presented the processes of finding similarity which we used as our baseline model: given a text of length n, a pattern of length m, and a maximal number of errors allowed, k, we want to find all text positions where the pattern matches the text up to k errors. Errors can be substituting, deleting, or inserting a character. The solutions to this problem differ if the algorithm has to be on-line (that is, the text is not known in advance) or off-line (the text can be preprocessed). In our research, we operated off-line algorithms.

Another widely-used approach to finding all the pairs in datasets that correspond to the same real-world entity is Natural Language Processing (NLP). Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words. In NLP applications, Murty (2003) claimed that even though traditional approximate string match algorithms may be applicable here, the lexicon will be searched more than once. One problem with these algorithms is that their hardware implementations are not simple. Murty (2003) proposed the solution in his paper: a dedicated co-processor for string matching that uses memory interleaving and parallel processing techniques can relieve the host CPU from this burden. In our research, we also noticed the time-consuming problem for NLP programming and tried different methods to speed it up, for example, the application of the awesome_cossim_topn function.

2.2 Data Collection and Preprocessing

2.2.1 Data Description

We have four datasets named Website_Txt, xp, bq and bq_financial. For the dataset Website_Txt and xp, they both have all non-null values on the company's domain name. Bq and bq_financial are one-to-one data by company id. So, we can join Website_Txt and xp by company domain name and join bq and bq_financial by company id. Xp and bq are data from two different vendors. They both include the domain name of companies, but there is some missing information on it. Additionally, they both contain all values on the company_name column, which includes numbers and strings, so we consider using a fuzzy match to join the company names. However, we find that datasets, xp and bq, include some empty strings in the company's name, company_name_clean, and company_name_join. Therefore, we operate some data preprocessing for the company's name to replace the company_name_clean and company_name_join.

2.2.2 Data preprocessing

For the company_name column, we unify its form by capitalizing the first letter of each word. For company_name_clean, firstly, we convert the company_name to lowercase. Next, we strip and remove punctuation, and do a deep clean by deleting the company suffix, such as "lnc", "llc" and "inc" to get it. As for the company_name in number form, we just make company_name_clean equal to it. Finally, we remove the whitespace in company_name_clean to get company_name_join. For example, the original company's name is 'Cavalry Management Group LLC', after data preprocessing, company_name is 'Cavalry Management Group Llc', company_name_clean is 'cavalry management group' and company_name_join is ‘cavalrymanagementgroup’.

It deserves to mention that in ranking models, we go through the same data preprocessing as above except the deep clean by deleting the “lnc”, “llc”, and “inc”. For information about city and state, we combined them and created a new column called “Address” to store it. Furthermore, we also strip and remove punctuation for Sci2_desc, which is the information about industry types.

After data preprocessing, we unify and improve the form of the company name. Additionally, we complete vacant strings for company_name_clean and company_name_join columns in datasets xp and bq. By clean data of the company name, industry, city, and state, we not only beautify the dataset visualization in the APP but also make the approximate matching models easier to find the correct matching pair.

2.3 Approximate String Matching Models

2.3.1 Modeling Workflow

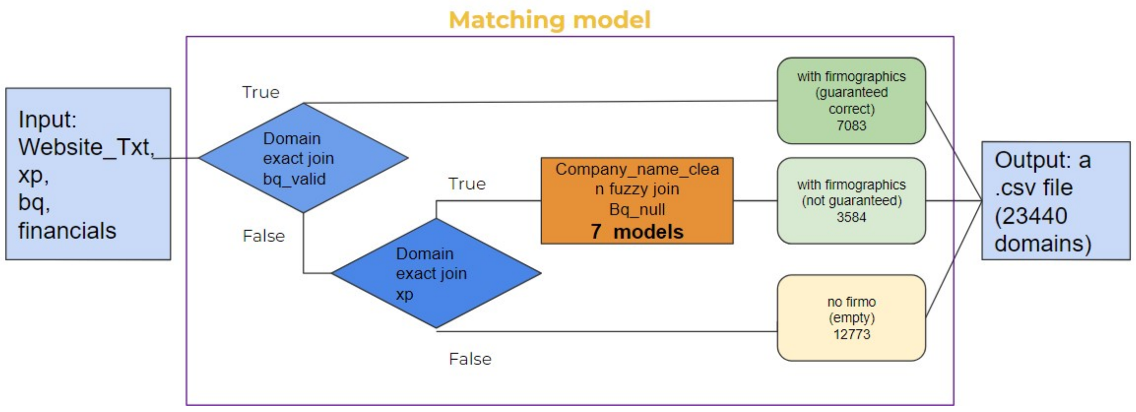

Matching model
Input: Website_Txt, xp, bq, financials
True
Domain exact join bq_valid
False
True
Domain exact join xp
False
Company_name_clean fuzzy join Bq_null
7 models
with firmographics (guaranteed correct) 7083
with firmographics (not guaranteed) 3584
no firmo (empty) 12773
Output: a .csv file (23440 domains)

Figure 1: Matching Algorithm Workflow

In this matching algorithm model part, our group uses multiple fuzzy matching methods such as TF-IDF, Word2Vec, SBERT, Databolt smart join, and ranking models. Even though we use different ways, our general framework is similar. The figure above shows how our matching model works.

We input four preprocessed datasets, Website_Txt, xp, bq, and bq-financial, into the matching algorithm. Firstly, we split Website_Txt into two datasets, Website_Txt_0 and Website_Txt_1, by whether domain_name of Website_Txt exactly joins bq_valid. Website_Txt_0 is a dataset that domain name of Website_Txt exactly matches bq_valid. Then we merge Website_Txt_0 with the dataset bq_financial on id and get the comprehensive dataset that corporates financial data we need. This is a sub-dataset of the final result, named With_Firmographics.

Next, we do the same process with Website_Txt_1 and xp. Split Website_Txt_1 into two datasets by whether we can merge with the xp dataset by the domain name. We exact match the part of Website_Txt_1 and xp and name it Website_Txt_2. Then we do fuzzy matching this dataset with the bq dataset. we get one dataset, With_Firmographic_Not_guaranteed.

Our main goal is to accurately fuzzy match two datasets using company information in order to broaden our coverage and provide users with comprehensive information about companies, including firmographic and financial information. We will compare many different models, including Databolt Smart Join, TF-IDF, Word2Vec, Sentence-BERT(SBERT), and Ranking models.

Lastly, we output the other part of Website_Txt_1 and xp that is not merged as No_Firm. We have no way to add any information about these companies to this part of the data.

We combine three sub-dataset and make it a csv file, helping the app takes less time when app users search for certain companies.

2.3.2 Baseline Model: Standard Fuzzy Matching

In order to compare the performance of our models, we need a baseline model. We choose the Databolt smart join algorithm (Databolt, 2020), which is simple and provides a fuzzy join function. The smart join algorithm will be our baseline model and an accuracy score from the model will act as a reference in our project.

In the baseline model, we calculated the distance between company names to find the top five similar companies for each company. We built two baseline models using two popular gap penalties for strings: Affine gap and Levenshtein distance, and both can be easily computed in python.

Affine gap Affine gap penalty is the most widely used gap penalty function. Affine gap penalty is written as $A + B \cdot (L-1)$, where L is the length of the gap, A is the gap opening penalty, and B is the gap extension penalty. We can consider the gap opening penalty as charging for the first gap symbol, and the gap extension penalty as charging for each subsequent symbol added to the gap (Rosalind, 2012).

Levenshtein Levenshtein distance is a string metric to measure the difference between two sequences, which is the minimum number of single-character edits required to change one word into the other. it is useful for suggesting spelling corrections and detecting plagiarism. The higher the distance, the more different the two strings are. For example, the Levenshtein distance between ”kitten” and ”sitting” is 3, since the following 3 edits change one into the other, and there is no way to do it with fewer than 3 edits: edit “kitten” to “sitten”, “sitten” to “sittin”, and “sittin” to “sitting” (add “g” at the end). An “edit” is defined by either insertion of a character, a deletion of a character, or a replacement of a character (Nam, 2019).

2.4 NLP Embedding Models

NLP applies embedding techniques to realize fuzzy matches and treat similarity scores as closeness scores. We can define the lowest bound score to determine if two items are matched. We developed three models using NLP word embedding techniques, Word2Vec, Ngram TF-IDF, and SBERT, which embed company name data and compute cosine similarity distance to fuzzy match companies. The three models are single-feature models, only using company names from two data vendors, xp and bq.

2.4.1 N-gram & TF-IDF

N-gram usually means a sequence of N words. For example, “research result” is a 2-gram. However, since we operate a matching algorithm according to company names, we change N-gram to N letters. The number of letters is set to 3 for the “company_name_join” column both in xp and bq files. The combination of N-gram and TF-IDF tables can function well for string matching problems. TF-IDF Vectorizer is a measure of comparing the number of times a word appears in a document with the number of documents the word appears in. We establish a TF-IDF vectorizer table to encode the N-gram results of company names. Next, we implement the awesome_cossim_topn function from the sparse_dot_topn library to calculate the cosine similarity of company names in different files. The awesome_cossim_topn function is good at dealing with sparse large datasets, which conform to the feature of the TF-IDF vectorized table of N-gram. As a result, by combining methods of Ngram, TF-IDF, and awesome_cossim_topn function, we can speed up the strings' matching process. For example, the speed for 26573*334256 data is only ten seconds. But what we can't ignore is that this method is poor at matching company names that their lengths are less than the settled number of letters of N-gram, which is 3 here.

2.4.2 Word2Vec

Using a neural network with only a couple of layers, Word2vec tries to learn relationships between words and embeds them in a lower-dimensional vector space. Word2vec trains words against other words that neighbor them in the input corpus, capturing some of the meaning in the sequence of words.

Since Word2vec models usually learn relationships between words, we need to improve this model and make this model learn relationships between company_name(phrase). Using a for loop, we tokenize each company_name_clean row. After creating the corpus, generate the word vectors by passing the corpus through word2vec.

After generating the word vectors, We start to filter the similarity of company_name_clean in the two datasets. Since we use matrix operations in this part, the actual running time of this part is very short. In about a few seconds, we can get the highest similarity data in the bq dataset corresponding to each data point in xp.

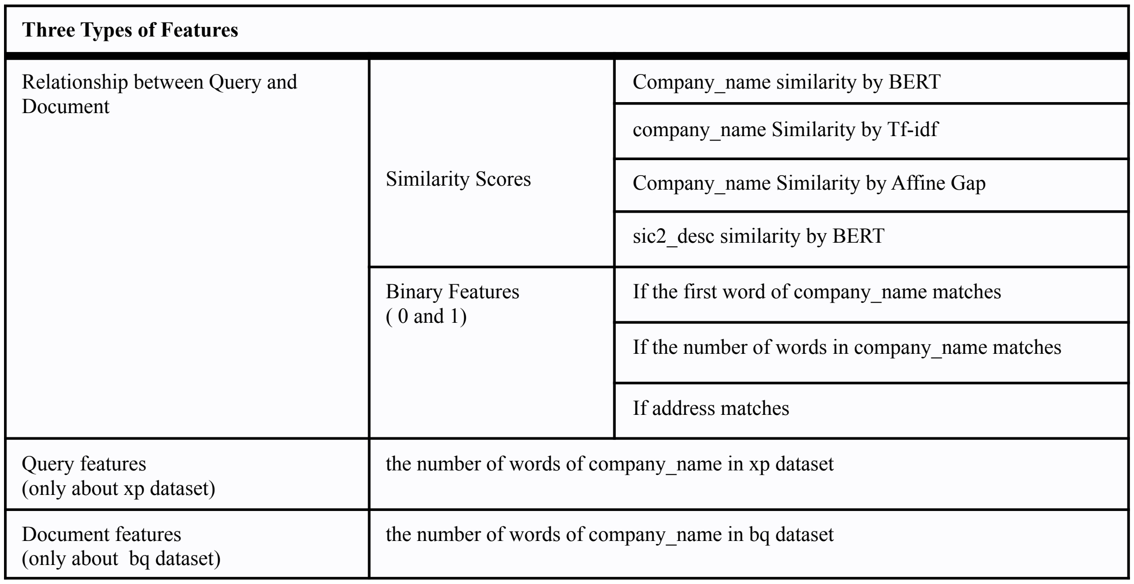
2.4.3 Sentence-BERT (SBERT)

We use one of the pre-trained SBERT models (msmarco-MiniLM-L6-cos-v5) to do data embedding on the cleaned company name and calculate the cosine similarities, with a CUDA device. When the similarity is greater than a certain threshold, we predict the corresponding record with the highest similarity in xp.

2.5 Ranking Models

Ranking is a subset of supervised machine learning, which takes a set of data points, and a query, and ranks the data points by giving a relevance score, indicating how relevant the data point is in the current group (Simon, 2021). In our project, each company in the xp dataset is a query and the companies in the bq dataset without the company's domain name are a set of data points. We use ranking models to find the most relevant result in the bq dataset for each company in the xp dataset.

2.5.1 Feature Engineering



Three Types of Features		
Relationship between Query and Document	Similarity Scores	Company_name similarity by BERT
		company_name Similarity by Tf-idf
		Company_name Similarity by Affine Gap
		sic2_desc similarity by BERT
	Binary Features (0 and 1)	If the first word of company_name matches
		If the number of words in company_name matches
		If address matches
Query features (only about xp dataset)	the number of words of company_name in xp dataset	
Document features (only about bq dataset)	the number of words of company_name in bq dataset	

Figure 2: Three Types of Features

Being different from baseline and NLP models, feature engineering plays a crucial role in ranking models. The features for ranking models can be classified into three types: the information only given by query (xp dataset), the information only given by document (bq dataset), and the relationship between query and document, respectively. The feature created by the relationship between query and document includes two parts. One is similarity scores, having four features shown in the following figure. For the information about the company name and industry type(sic2_desc) in the query and document, we get four similarity scores by three different models mentioned before: word2vec, SBERT and Databolt smart join with Affine gap penalty. Others are binary features, having three features. We set it to 1 if the condition below is matched and 0 otherwise. Additionally, we have one query feature - the number of words of company_name in xp dataset, and one document feature - the number of words of company_name in xp dataset.

2.5.2 Feature Importance

We build our models with LightGBM and XGBoost rankers. For the LightGBM model, we use lambdarank as an objective function since lambdarank is very effective on the optimizing ranking function, NDCG score (Tamara,2020). For the XGBoost model, we use the NDCG score as an objective function.

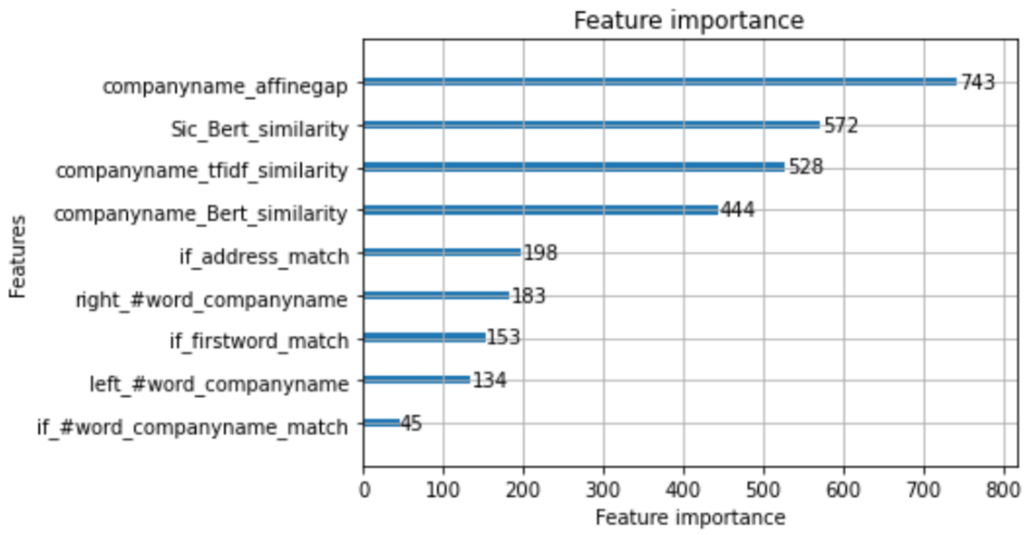


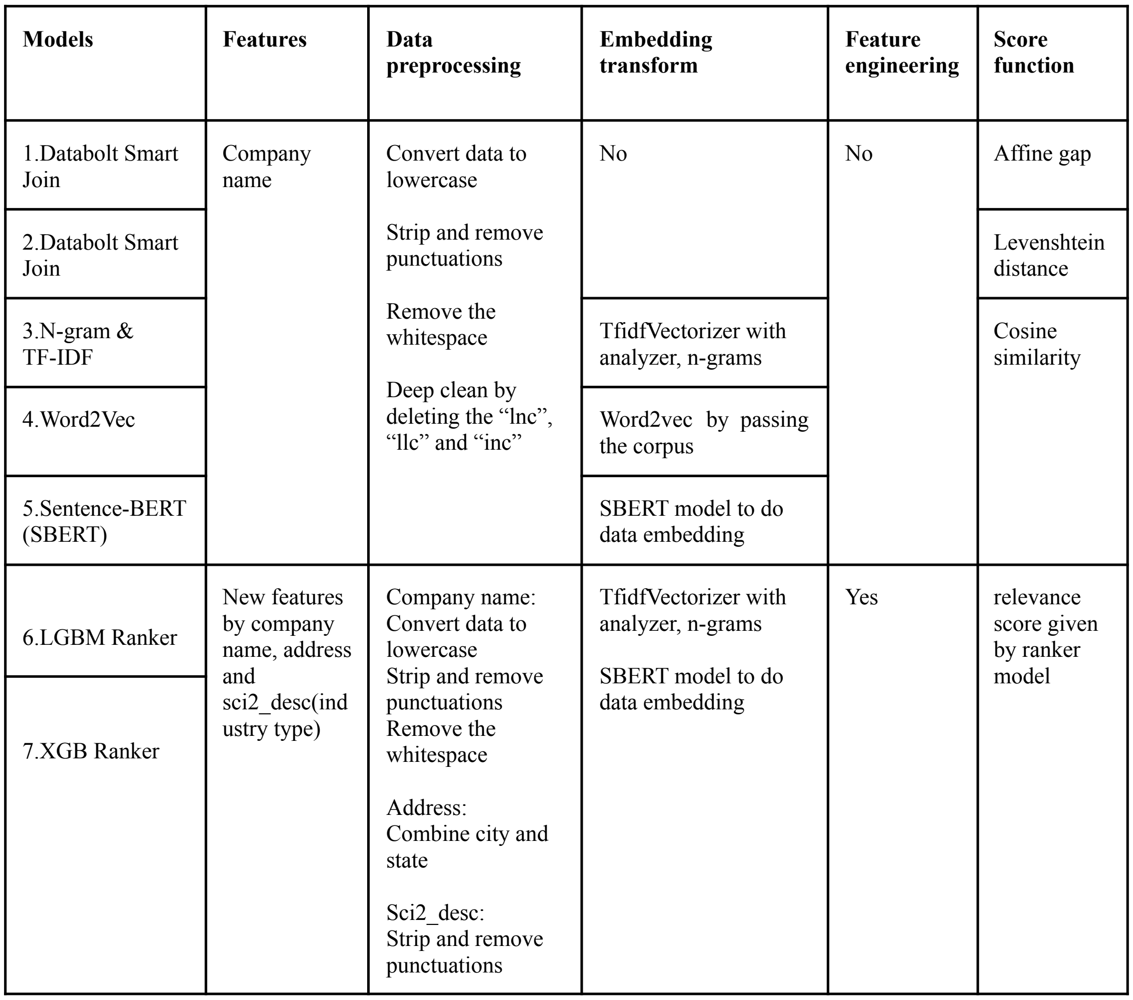
Figure 3: The Importance of Features

The graph above shows the importance of different features in the LightGBM model. Similarity scores are important features. Among them, the affine gap of company name has the most crucial influence on improving NDCG scores. Besides, compared to baseline and embedding models, the information about address and industry type(sic_desc) also can be utilized and contributes to the improvement of the fuzzy matching result. Finally, we find the LightGBM model has better performance.

2.6 Comparison of All 7 Models

Our group compares all 7 models from five different points of view: features, data preprocessing, embedding transform, feature engineering, and score function. The general idea for matching models is that we do data preprocessing, use models to transform data or create features by feature engineering, and finally use scores calculated by function to find the most relevant result.

As is shown in the table below, the first five models only have one feature, company name, and another two ranking models have new features by feature engineering on information about company name, address and industry. For the first five models, we convert the company name to lowercase, strip and remove punctuations, remove the whitespace, and deep clean by deleting the “lnc”, “llc” and “inc” for data preprocessing.



Models	Features	Data preprocessing	Embedding transform	Feature engineering	Score function
1.Databolt Smart Join	Company name	Convert data to lowercase Strip and remove punctuations Remove the whitespace Deep clean by deleting the "lnc", "llc" and "inc"	No	No	Affine gap
2.Databolt Smart Join					Levenshtein distance
3.N-gram & TF-IDF			TfidfVectorizer with analyzer, n-grams		Cosine similarity
4.Word2Vec			Word2vec by passing the corpus		
5.Sentence-BERT (SBERT)			SBERT model to do data embedding		
6.LGBM Ranker	New features by company name, address and sci2_desc(industry type)	Company name: Convert data to lowercase Strip and remove punctuations Remove the whitespace Address: Combine city and state Sci2_desc: Strip and remove punctuations	TfidfVectorizer with analyzer, n-grams SBERT model to do data embedding	Yes	relevance score given by ranker model
7.XGB Ranker					

Figure 4: Comparison of All Models

For ranking models, we did not do deep clean because we compare the NDCG score and find NDCG score is higher for ranking models without deep clean. For address and sci2_desc(industry type), we also do some simple data preprocessing. For embedding transform, we use Tfidf Vectorizer with analyzer n-grams to transform data in the third model. For the fourth model, we transformed data into word vectors by passing the corpus trained by the word2vec. For the fifth model, we use the SBERT model to do data embedding. For ranking models, we also use some of the same ways to transform data. But we only do feature engineering for ranking models. For the score function, the Databolt fuzzy matching uses affine gap and Levenshtein distance, respectively. A lower score means a closer result. The three models, N-gram & TF-IDF, Word2Vec, SBERT, use cosine similarity. For ranking models, we use relevance scores.

2.7 Evaluation Metrics for Fuzzy Matching

Using adequate metrics and understanding them are one of the most important tasks in machine learning. In our matching process, our goal is to accurately match companies from two vendors as much as we can by using company information. In the fuzzy join part, we will find and match the most similar company by using many features including company names or industry types. It works like a search engine that outputs a list of items.

To evaluate recommender systems, we need to consider how relevant the results are and how good the ordering is. Typical classification and regression metrics measure whether the predicted value is close to the actual value. But they do not account for the order of predictions. For example, the f1 score is a commonly used evaluation metric for the classification machine learning models. However, since it does not consider the order of predictions, it cannot measure how well a recommender system works.

We chose Normalized Discounted Cumulative Gain(NDCG) score as our evaluation metric, which is a popular measure of ranking quality and is often used to measure the effectiveness of web search engine algorithms. We will find the best-performing model based on NDCG@5 scores, a method that only considers the top five recommendations for each company data from a model and calculate an NDCG score.

2.8 Results and Conclusions

We used three different types of models for fuzzy matching, Databolt smart join methods, models using NLP embedding techniques, and ranker models that used several features. In the bar chart below, we compared ndcg scores of models. Ndcg score is between 0 and 1, and the greater the ndcg, the greater the relevance of the recommended results.

Databolt smart join methods are our baseline model and two scores of the baseline models, 0.74 and 0.65, are standards for evaluating other models' performance. Although they showed better ndcg scores than we expected, it took much more time to run it than other methods. Since we will be dealing with many features of over 300,000 companies from two data vendors, we have to consider the time efficiency of models. When considering both ndcg score and execution time, our baseline model cannot be said to be the best model.

NLP embedding models took far less time than the baseline model and performed well. Sentence Bert methods showed 0.78, the best score among three NLP embedding models.

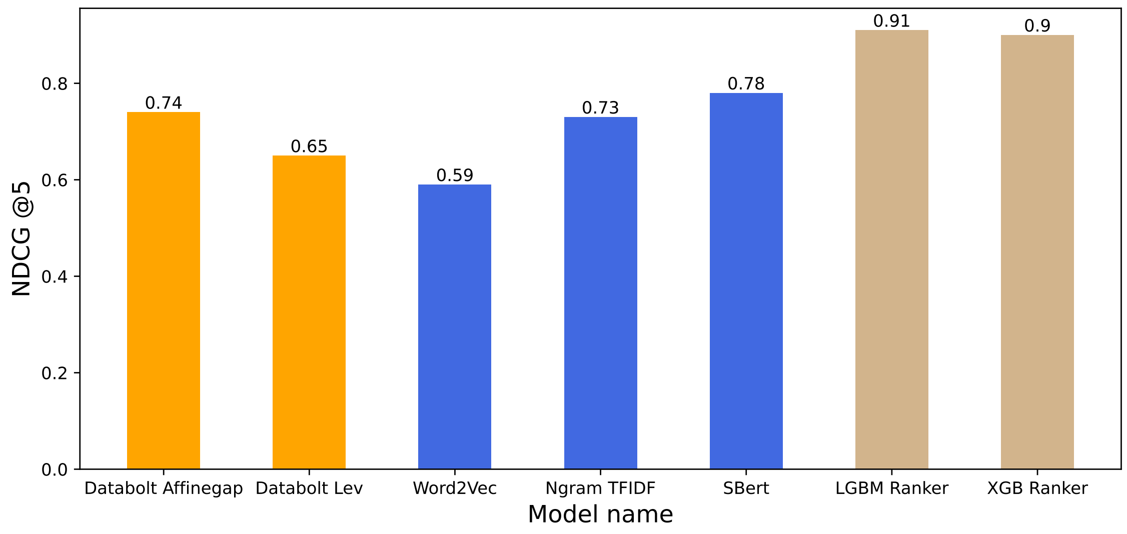


Figure 5: NDCG scores of Fuzzy matching models

While it is a single feature model, it showed better performance and also was time efficient than the Databolt smart join model.

Our two ranker models, Light GBM and XGBoost ranker model showed the highest results, over 0,9. In ranker models, we added more features including address and industry types of companies. We concluded adding features led to a significantly higher ndcg score than other models and chose the LGBM ranker model and use it for the fuzzy matching part in the matching algorithm.

2.9 Applications of Merged Data

This project aims to develop an App to help users find relevant companies to invest in a specific investment criterion. Our App provides users with a more personalized experience by offering several filter options and models and returning a small number of high potential companies for users to do investment research on. The whole project is split into four different tracks: NLP search algorithms, Predictive modeling, Industry tags, Keywords search, and Cloud. Our team focused on the predictive modeling part and also collaborated with students in other tracks to build the App.

In the collaborative process, our group mainly provides a combined data parquet file, including domain name, company name, address information, industry tags, and firmographic information. This added information is very important for our customers, since they can use this data to find relevant companies to invest in a specific investment criterion.

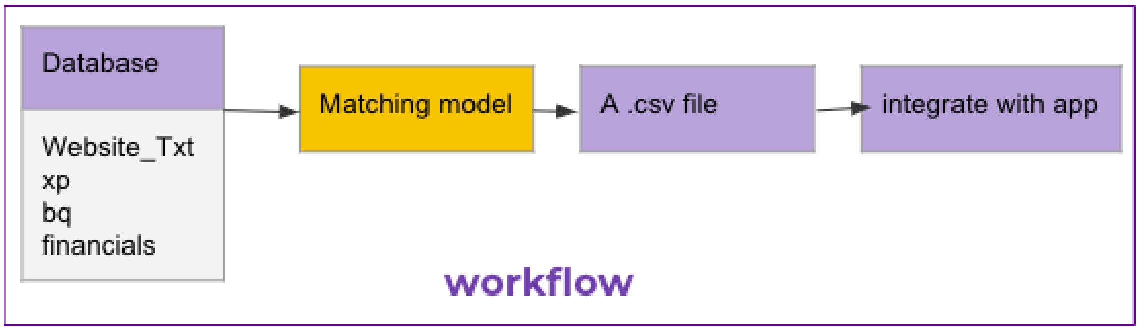


Figure 6: Application Workflow

The above figure shows our workflow. We get these data (Website_Txt, xp, bq, bq-financial) from our database. We input these data in our string matching model in order to add more information to these companies. Then we deliver the output CSV file to other groups. They can make sure our searching algorithm can search from these data, and put the result in the user interface. The App interface is attached in the appendix.

3 Finding Similar Companies by NLP

In order to find similar companies for all listed companies in the Website Txt file, we operated the TF-IDF embedding model on the content search column. Before we matched company names and it is more like word embedding, for this time, we dealt with paragraph contents and it is more like sentence embedding. The reason that we chose the content search column is that it does not have any missing values. From the point of view of the user, the content search column can best describe the most relevant content about the company. Before modeling, we cleaned the content search column by standard NLP pre-processing: removing stop words, lemmatizing, lower casing, and ignoring punctuations.

For the first version of our TF-IDF model, we found that although there are no missing values in the ‘content_search’ column, this column contains much invalid information, such as “something goes wrong”, “blocked by VPN”, and “have no access”. These cases occur because the vendor is blocked when it tries to obtain the website searching information. If several companies all have the same “something goes wrong” information, they will be recognized as similar companies with a similarity score of 1. However, they actually may not even be in the same industry. To avoid such wrong situations, our group deleted all data with invalid ‘content_search’ information.

TF-IDF embedder vectorized paragraphed string information. We search the most similar three companies in the same Website Txt file. A higher similarity score means matching is better. So, the most similar one will always be itself with a closeness score of 1. Our result table is presented as follows: the first "Domain Name" column is our query companies; the second column "Most Similar" is the company that has the highest similarity score except itself. The "Second Similar" is the second similar company. Since we don't have more data to evaluate the matching, we randomly choose some companies to check their websites to see if they are in the same industry. For example, the fourth observation is "koparibeauty.com". From its website, we can know it is a company selling natural skin care products. "nativecos.com" and "organicfiji.com" are also selling organic skin care products. Thus, from this perspective, the matching results are quite reasonable.

Table 1: Finding Similar Companies Outputs

	Domain Name	Most Similar	Second Similar
0	thrivemarket.com	healthylivingmarket.com	thrivesavings.com
1	madison-reed.com	hair.com	perfectlocks.com
2	beautycounter.com	ruecinq.com	astonishingskincare.com
3	koparibeauty.com	nativecos.com	organicfiji.com
4	letsdisco.com	kuraskin.com	shopkaike.com
5	thirteenlune.com	skinstore.com	b-glowing.com
6	born.com	themaintab.com	elementsbrands.com
7	wanderbeauty.com	donaldpliner.com	jacobsonfloral.com
8	standarddose.com	thefascination.com	thinkiam.com
9	ilmakiage.com	makeup.com	makeupgourmet.com
10	pytbeauty.com	redapplelipstick.com	personacosmetics.co
11	lovegoodly.com	vegancuts.com	thebridebox.com
12	savhera.com	cliganic.com	dharmaceuticals.com
13	futurederm.com	tattly.com	dfynt.com
14	mxt.co	kuraskin.com	tcosc.org
15	FreshMadeSkincare.com	sapeloskincare.com	letsdisco.com
16	vixxenn.com	angelshaveclub.com	angellaunch.com
17	travelbeauty.com	sapeloskincare.com	publicbeauty.com
18	DwightFunding.com	smallbusinessfunding.com	ivylender.com
19	nyxcosmetics.com	personacosmetics.co	lipbalmproducts.com

4 The CapRaise Model

4.1 Introduction to the CapRaise Score

CapRaise score is a behavioral signal developed by Fintent, a fintech startup. It is produced by around 600,000 companies in the United States matching the Series A/B/C+ Growth profile (Fintent, n.d.). The score is designed to help VC and PE firms discover more companies that match their investment profile and are looking for new sources of financing. The signal is delivered monthly in the form of 100-point scale numeric data.

In addition to the CapRaise scores, there are several other features we can make use of. Firstly, all records are categorized into six topics, namely: capital injection, crowdfunding, preferred stock, recapitalization, venture capital, and virtual data room. Secondly, we have the date information of each specific fund-raising activity. Furthermore, we are provided with web ratings scores based on website activities of each firm.

Our objective is to combine the CapRaise scores, web activity ratings, and fundraising dates and topics, to build predictive models for capital raises. The labels to predict are created based on whether these companies will raise more funds in a latter period of time.

4.2 Data Cleaning and Preprocessing

The original data consists of correlated time series which are not suitable for the models to take in directly. Therefore, it is necessary for us to preprocess the data and convert it to cross-sectional data before we use them to make predictions.

First of all, we need to take care of the date time information. This is particularly important because the labels are marked on different days for each company. An arbitrary use of fund raising dates will cause look-ahead bias and other forms of data leakage, which results in useless models. To prevent such problems, we first take the six-month window before the event date for each company, and then calculate the gap between each fund-raising activity and the prediction date.

For each company, the records are aggregated with respect to the six fundraising topics. Within each topic group, we calculate the average CapRaise scores, Web activity ratings, and time gaps. The final processed dataset therefore consists of 18 features (6 topic groups, 3 features for each group). Missing values are filled with the default level of each feature.

4.3 Models and Empirical Results

We test three major categories of models. Firstly, we use LightGBM as a baseline model. Then we experiment with three interpretable models (ReLU-DNN, GAMI-Net, and EBM) from the PiML toolbox. Finally, AutoML (Auto Machine Learning) models are used to carry out automatic searches over a variety of different models. We will introduce these models and compare their respective performances in the following sections.

4.3.1 The baseline model

The baseline model is the LightGBM Classifier, a classical model using tree based learning algorithms. As the size of our data is not very large, we set the max depth to 3 and number of estimators to 50 to prevent overfitting. We also implemented balanced training to avoid class imbalance biases.

4.3.2 The PiML models

In this section we test three interpretable machine learning models from the PiML toolbox: ReLU-DNN (Deep Neural Networks with ReLU activation functions) and GAMI-Net (Generalized Additive Model with Structured Interactions) are multi-layer network models capable of handling thousands of parameters. EBM (Explainable Boosting Machine), on the other hand, is a tree based interpretable model featuring relatively fewer parameters.

Starting with the default settings, we carry out hyperparameter tuning based on the cross-validation performances. Since we found that the neural networks models seem to be susceptible to overfitting, we decreased the number of nodes in the networks and applied more severe penalty terms. The EBM model, by comparison, shows more consistent performances on training and validation sets, so we just slightly tweaked the number of interactions to obtain optimal F1 scores.

4.3.3 The AutoML model

Automated machine learning (AutoML) has been a trending topic in the past few years. Several powerful open-source packages have been developed to automate the process of applying machine learning to real-world problems. These AutoML models cover every stage of the machine learning process, including feature engineering, model selection, and hyperparameter tuning. All we need to do is to designate parameters for the AutoML algorithm itself. In our case, we set the maximum model selection time to 5 hours and use cross validation (CV) as the resampling strategy.

4.3.4 Performance comparison

For performance comparison, we use the 5-fold cross validated F1 score. The reason to use the F1 score is that, rather than focusing on the accuracy, our model should emphasize on the balancing between precision and recall scores. After all, our goal is not just to predict more samples correctly, but to invest in more profitable companies. A higher precision score, in our case, suggests that we will waste less money on companies that are not worth investing in. The recall score, on the other hand, measures how many of all potentially lucrative opportunities are we able to capture. Therefore, the harmonic average of precision and recall, that is, the F1 score, is a more valid measure for model comparison.

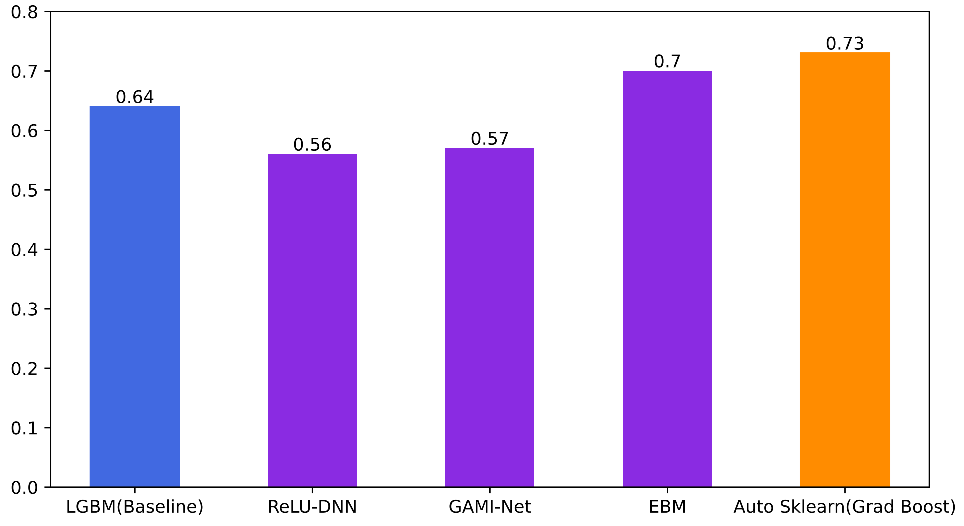


Figure 7: 5-fold Cross-Validated F1 Scores (Average)

As we can see from the bar chart above, the baseline LightGBM model actually has a solid performance, out-scoring the two neural network models. The EBM model from the PiML toolbox slightly out performs the baseline model. The best performance, however, goes to the gradient boosting model selected by the AutoML algorithm, with the highest F1 score of 0.73. Based on this result, we can conclude that decision tree based models are generally more suitable for the classification task in our case.

4.4 Model Explanation

To explain how the classifier actually makes a decision, we draw the Shapley Additive Explanations (SHAP) plot based on samples in the holdout dataset. The SHAP values measure the impact of each feature based on cooperative game theory, which explains how the features influence the decision-making process of the machine learning model.

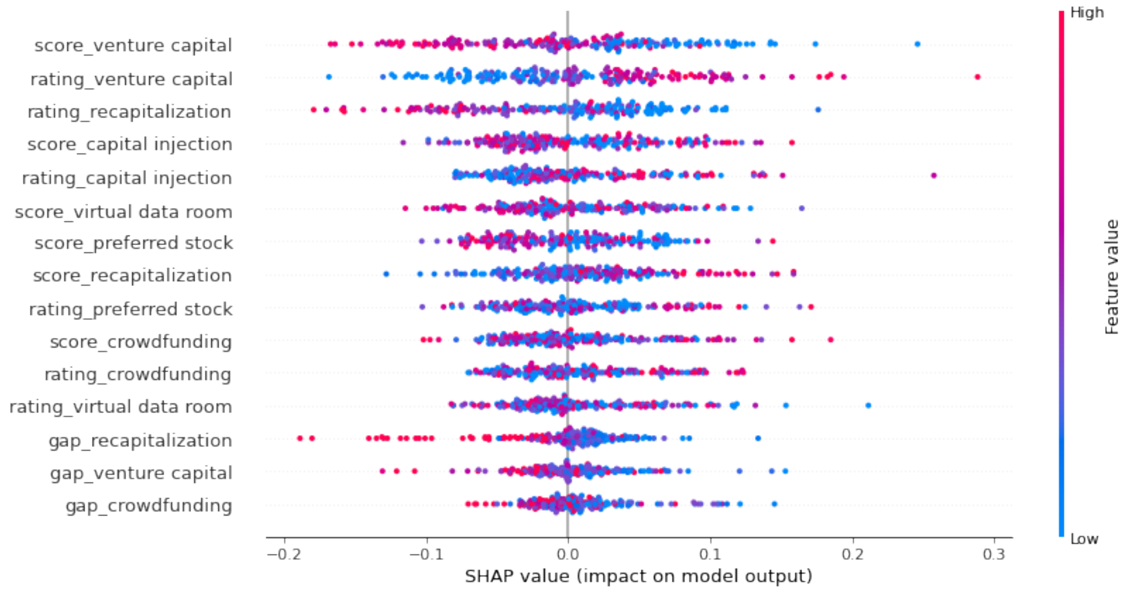


Figure 8: Global SHAP of the AutoML model, performed on Holdout Data

We have three main comments on this result. Firstly, the average CapRaise score of the Venture Capital group appears to be the most important predictor. This indicates that companies with high quality venture capital activities in the past six months are more likely to be raising additional funds in the future.

Secondly, Web Activity Ratings generally have less impact than the CapRaise scores. Even if a company has decent fund raising records and promising potentials, this is not always reflected in the activities of its website. This phenomenon not only appears in this particular case, but also in all the other models we have tested in this study.

Our last comment is that the gap of fund-raising activities does not serve as a good predictor for future profits. The gap features are consistently ranked last in all the models we have tested in terms of SHAP values.

5 Conclusions and Future Work

In conclusion, our team has effectively completed the task of building predictive models. First of all, our NLP fuzzy matching models improved the coverage of matching company information from different vendors. Next, similar company search algorithms grouped similar companies by companies' website text, helping app users who want to invest in multiple companies in a similar industry. Lastly, the CapRaise model has demonstrated significantly higher performances than the baseline model, aiding investors to predict if companies are raising money based on information, for example, time series data of fund raising activities and web ratings of companies.

There are, however, several further improvements we could make to the models. For the NLP models in the fuzzy matching part, namely the TF-IDF and SBERT models, we were using the general versions directly. If we were provided with more computing power, we might be able to improve the result by fine-tuning the parameters of these models. Furthermore, the effectiveness of the CapRaise model is slightly handicaped by the relatively small size of the data set. It remains to be examined whether the strong performance of this model could be replicated in real-world applications.

References

Baeza-Yates, R., & Navarro, G. (2006). A faster algorithm for approximate string matching. In *Combinatorial Pattern Matching: 7th Annual Symposium, CPM '96, Laguna Beach, California, June 10-12, 1996. Proceedings* (pp. 1-23). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-61258-0_1

Cucumides, T. A. (2020). *Learning-to-rank with LightGBM (Code example in python)*. Tamara Alexandra Cucumides. Retrieved August 15, 2022, from https://tamaracucumides.medium.com/learning-to-rank-with-lightgbm-code-example-in-python-843bd7b44574

Databolt. (2020). *d6t/d6tjoin: Fuzzy joins for python pandas - easily join different datasets*. GitHub. Retrieved August 15, 2022, from https://github.com/d6t/d6tjoin

Fintent. (n.d.). *Backtested CapRaise Score*. Fintent. Retrieved August 15, 2022, from https://fintent.net/cap-raise-score/

Hall, P. A. V., & Dowling, G. R. (1980). Approximate String Matching. *ACM Computing Surveys*, *12*(4), 381–402. https://doi.org/10.1145/356827.356830

Kalyanathaya, K. P., Akila, D., & Suseendren, G. (2019). A Fuzzy Approach to Approximate String Matching for Text Retrieval in NLP. *Journal of Computational Information Systems*, *15*(3), 26-32.

Lind, S. (2021). *Learning to Rank using XGBoost. sci-kit learn and Pandas | by Simon Lind | Predictly on Tech*. Medium. https://medium.com/predictly-on-tech/learning-to-rank-using-xgboost-83de0166229d

Murty, V.S., Raj, P.C. R., & Raman, S. (2003). Design of a high speed string matching co-processor for NLP. In *16th International Conference on VLSI Design,* (pp. 183-188). IEEE. 10.1109/ICVD.2003.1183134

Nam, E. (2019). *Understanding the Levenshtein Distance Equation for Beginners | by Ethan Nam*. Medium. Retrieved August 15, 2022, from https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0

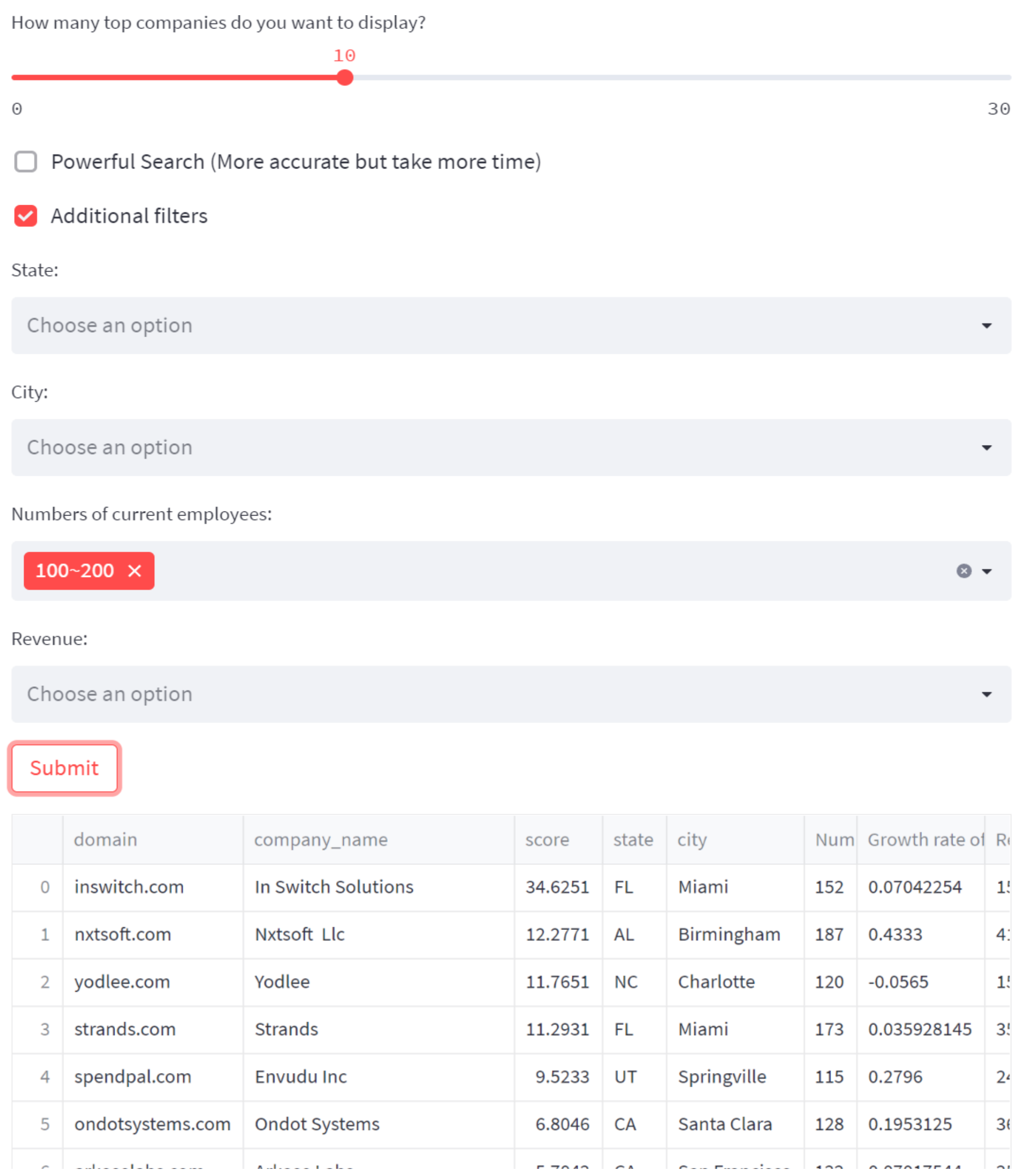
Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys*, *33*(1), 31-88. https://doi.org/10.1145/375360.375365

Rosalind. (2012). *ROSALIND | Global Alignment with Scoring Matrix and Affine Gap Penalty*. Rosalind. Retrieved August 10, 2022, from https://rosalind.info/problems/gaff/

XGBoost. (2021). *Python API Reference — xgboost 1.6.1 documentation*. XGBoost Documentation. https://xgboost.readthedocs.io/en/stable/python/python_api.html

Appendix

This figure is an example of our APP.



How many top companies do you want to display?

10

0 30

Powerful Search (More accurate but take more time)

Additional filters

State:

Choose an option

City:

Choose an option

Numbers of current employees:

100~200

Revenue:

Choose an option

Submit

	domain	company_name	score	state	city	Num	Growth rate of	R
0	inswitch.com	In Switch Solutions	34.6251	FL	Miami	152	0.07042254	1
1	nxtsoft.com	Nxtsoft Llc	12.2771	AL	Birmingham	187	0.4333	4
2	yodlee.com	Yodlee	11.7651	NC	Charlotte	120	-0.0565	1
3	strands.com	Strands	11.2931	FL	Miami	173	0.035928145	3
4	spendpal.com	Envudu Inc	9.5233	UT	Springville	115	0.2796	2
5	ondotsystems.com	Ondot Systems	6.8046	CA	Santa Clara	128	0.1953125	3
6	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]