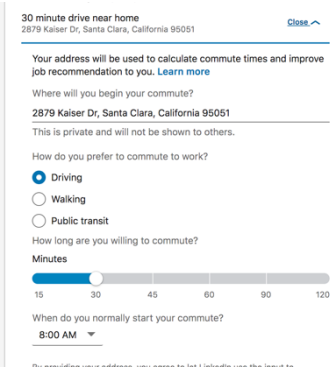
Discover LinkedIn Job Seeker's Commute Preference

Dianxia Yang
dianxiay@stanford.edu

***Abstract*—LinkedIn is always working on connecting the best job opportunities to our valued members. Based on user research, commute had been one of the major factors when members look for their dream jobs. This report is trying to resolve two sub-problems: (1) Find out whether a LinkedIn member is willing to provide their commute information to connect to opportunities. (2) What is the member's expected maximum commute time. With these problems resolved, we could either make precise promotion to our members or leverage this implicit inferred data for job recommendation. I leveraged classification machine learning algorithms to both problems with same feature set respectively. It turns out that I was able to obtain decent prediction on problem (1), but the result for problem (2) is not as good as expected.**

I. Introduction

I am current a full-time software engineer in LinkedIn Jobs Team and I'd like to work on a project related to my daily work. This year we've rolled out a feature to collect jobseeker's commute preference. A member can provide his exact address and preferences to commute, e.g. driving v.s. public transit, the time range of the commute, as illustrated on the screenshot here. For members who provided commute preference, we are able to show member jobs within their commute preference range and recommend jobs based on their commute preference.



Note that all commute preferences we collected from members today *are organically input by member*. In order for us to provide more value to our job seeker from this feature, we want to create a better coverage on LinkedIn members with commute preference. After research, I've decided to scope my project to the following two sub problems:

- Predict if a user is a commute sensitive user and is willing to put commute preference in careers interest page. We will reference as *Willingness Problem* in the rest of this paper.
- Predict member's expected maximum commute duration range by driving, falling into different categories in minutes like l5min, 60 min, etc. We will reference this as *Duration Problem* in the rest of this paper.

With these two problems resolved, we will be able to leverage this data for precise feature promotion, e.g. a popup box saying "Based on your activity you seems to be interested in jobs near you, would you like to provide us your detailed commute preference?", or, more directly, we can leverage this predicted implicit data to improve job search and recommendation directly.

The input of our project would be LinkedIn member's basic information and job seeking related activity tracking data. The output of our project will be a probability score in Willingness problem and will be a list of score indicating the likelihood of the commute duration falling into different duration categories.

As a side note, since I am dealing with LinkedIn's industry data and product, please understand that I will hide some information for integrity whenever there is a need, including dataset size, some data distributions, etc.

II. Related Work

Unfortunately, the problem I am trying to resolve in this project is extremely specific to our product so there are no direct researches that I could leverage. But it turns out that I got some inspirations on few other job seeker related studies within company done by other people. The information I leveraged from those studies includes, their usage of features and data source, the models they are leveraging as well as the ways they are improving the model, which I cannot give more details in this paper.

III. Dataset and Features

This is an industry level project where I am dealing with big data generated from our production. I'd like to underscore that there is *significant amount* of data engineering effort involved to make up the dataset ready for training. Since I am not able to share the code for this part due to privacy issue, I will describe into details about my step to generate and selecting features.

A. Datasets

All raw datasets I used for this project are persisted in HDFS, which stores our ETL'ed database data as well as our tracking events sent from Kafka. For this project, here are the datasets I leveraged:

- Standardized member profile data, which includes member's derived location.
- Standardized job posting data, which includes job postings' derived location.
- Member's job activity tracking events, including job view/search/apply/save.
- Member's careers preference data, which includes member provided preferred location and activity level.
- Standardized geo data, which provides geo coordinate information (latitude/longitude) given a geo location.
- Existing commute preference data organically provided by members.

All data I used for this project had been pre-sanitized with our data team and had obfuscated all personally identifiable information.

The datasets I selected for this project is mostly coming from my product intuition coming from my day-to-day work, as well as referencing to some similar research done within the company.

B. Label Preparation

1) Willingness Problem Labeling: It is obvious that the positive data could be generated from our existing commute preference data, which we simply mark all members inside the data set as positive.

The negative data, however, is something we need to figure out how to build by ourselves. We cannot use all members who did not provide the information as negative, as a lot of them are simply not actively looking for the job and thus do not know about the feature. Based on the product intuition, I decided to generate negative labels for members who had **viewed the commute preference page but did not provide us information**. This approach makes sense in the way that if we are going to promote the feature with popup box or notification, the negative data described here actually simulated the situation when member become aware of this feature, and thus I believed it could capture our testing scenario.

2) Duration Problem Labeling: We are currently storing categorized commute duration for members like (15, 30, 45, 60, 90, 120). I believe it is hard to capture member's commute duration preference in 15-minute granularity, so I've remapped the label into: [15, 30] → 0, [45, 60] → 1, [90, 120] → 2 to reduce the number of classification goal.

C. Features & Generation Process

We can categorize the features into following two categories:

1) Member derived data: I've directly generated features of member industry and member job activity level from our derived data set.

2) Job Activity Distance Data: In general, This category of features is generated from our raw data in the following steps:

1. Retrieve all <memberId, jobId> pairs from job view/ job apply / job save tracking events.
2. Join and find out member's derived location from our dataset. Also join and find out the job's location.
3. Join the locations with our standardized geo dataset, get the standardized geo coordinates for members themselves and job location.
4. Calculate the distance from member to job with the following Haversine formula:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

5. Group the data with memberId and then build a member to vectors map, for example:
   <member_A → vector containing all job view distances for member A>
6. Since most models cannot deal with variant length vectors, we will derive 3 features from the vector, which is **count, average distance and standard deviation**.
7. Apply steps 1~6 to all job view, job apply, and job save data. Apply steps 3~6 to member provided preferred location data, and job search data (as the location is directly associated with member in these two datasets). As a result, we generated 15 features here.
8. Specially, for job search, I've excluded searches in country and state granularity as it's geo coordinate is kind of random in such a big geo entity. Instead, I've introduced two features: country_search_count and state_sarch_count to capture this information.
9. At the end, I standardized the numeric data into values with 0 mean and 1 standardized deviation with:

$$z = \frac{x - \mu}{\sigma}$$

D. Deal with Missing values

As you can imagine, there will be a lot of missing values with my dataset as it is a norm that a member may never applied or proactively searched for a job, etc. I believed that it is a tradeoff between the model accuracy and the usefulness of the model, as we will can also do the similar transformation for missing values in test data and enables the model to work for more people. I decided to deal with the missing value in the following steps:

1) Remove all data rows missing job view data: It does not make sense to predict for a member who never viewed a single job.

2) Populate job apply and job search average distance and standardized deviation with job view data if missing.

3) Populate job save and member preference location from the job apply data filled in step.

4) Generate a feature of is_{some action}_missing = {0,1} based on the missing data populating result

The above missing value populating algorithm is mostly based on product intuition. There may be some better way to populate these fields and I will discuss in the Future Work section.

But note that this step is not required for XGBoost algorithm I used, which will be covered in the Methods section.

E. Sample and Dataset Split

After all the preprocessed data above, I've sampled a big enough amount of data as our full dataset, and do a 80/20 split for training/dev split, while I am leveraging the dev set mainly for model tuning.

Besides, I've set up a separate dataset with latest data as test set to report performance of different models. This followed the intuition that we are predicting the future data with older data.

IV. Methods

For the two sub problems, I am using the same set of features as well as similar set of machine learning algorithm, but with different configurations.

A. Deal with Imbalanced Data for Duration Problem

Before I actually implement any machine learning algorithm, I noted that the label distribution of Willingness problem is reasonably balanced. The Duration Problem, however, the labeled data is highly inclined into label 0 and 1, and the model maybe skewed to predict 0 and 1 if I do not take some actions.

To resolve this issue, I calculated a class weight with the following equation:

$$w_j = \frac{n}{kn_j}$$

The intuition of this formula is to assign a higher weight for minority classes. I then assign this class weight to the training samples according to the training sample class, this actually transformed all my loss functions in our methods as a weighted loss, which effectively penalized errors on minority classes.

B. Logistic Regression/Softmax Regression with L2 regularizations

We implemented the binary model for our Willingness Problem, the logistic regression with L2 standardization optimized the following loss function for binary model:

$$l(\theta) = \sum_{i=1}^{m} y^{(i)} \log\left(h\left(x^{(i)}\right)\right) + (1 - y^{(i)}) \log\left(1 - h\left(x^{(i)}\right)\right) + \frac{\lambda}{2}\|w\|^2$$

$$h_\theta\left(x^{(i)}\right) = \frac{1}{1 + e^{-\theta^T x^{(i)}}}$$

For Duration Problem, we implemented the softmax which replaced the hypothesis as:

$$p(y = i|x;\theta) = \frac{e^{\theta_i^T x}}{\sum_{j=1}^{k} e^{\theta_j^T x}}$$

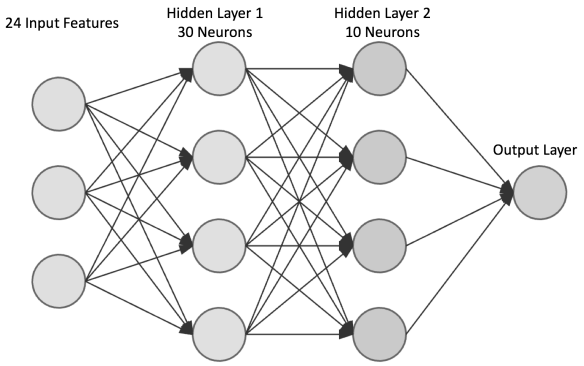
And the loss function to optimize becomes:

$$l(\theta) = \sum_{i=1}^{m} \log \prod_{l=1}^{k} \left(\frac{e^{\theta_i^T x}}{\sum_{j=1}^{k} e^{\theta_j^T x}} \right)^{1\{y^{(i)}=l\}} + \frac{\lambda}{2}\|w\|^2$$

C. Neutral Network

I've introduced neutral network to learn both problem with slightly different configurations.

For Willingness Problem, I've constructed a neutral network with 2 hidden layers with 30 and 10 neurons respectively. In both hidden layers, I am using ReLU activation function. And I used sigmoid function as our output layer because I would love to get the similar format of results I had from the previous logistic regression. I run the training process with 20 epochs. The network structure and forward propagation formula can be represented as:



$$Z_1 = XW_1 + b_1$$
$$A_1 = ReLU(Z_1)$$
$$Z_2 = A_1W_2 + b_2$$
$$A_2 = ReLU(Z_2)$$
$$Z_3 = A_2W_3 + b_3$$
$$\hat{y} = \sigma(Z_3)$$

For Duration Problem, I've constructed a neutral network with 2 hidden layers with 50 and 20 neurons respectively and are again using ReLU as hidden layer activation function. For output layer, I am using a softmax activation function which works for multinomial classification problem.

D. Decision Tree

I've shifted my focus to non-linear machine learning algorithms like decision tree.

I am using a Gini loss for my decision tree, defined as:

$$L_{gini} = \sum_k \hat{p}_c(1 - \hat{p}_c)$$

And the decision tree is constructed with the split rule which maximized the Information Gain (IG) in each split, which is defined as:

$$IG = L\left(R_{parent}\right) - L(R_{children})$$

I implemented decision tree for both Willingness Problem and Duration Problem. It turns out that the decision tree model tends to overfit, and after tuning the parameters I've set down using minimum samples pruning rule with 100 minimum sample for Willingness Problem, and 400 minimum sample for Duration Problem.

E. XGBoost

XGBoost stands for "Extreme Gradient Boosting" and has been a very popular decision tree ensembles algorithm used to learn structured data. The idea of XGBoost training is Additive Training. In each step, the algorithm fixes what have learnt and add one new tree at a time. We can represent the prediction at each step t as:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

To find the optimal tree to add in each step, intuitively we need to optimize our objective in each step:

$$obj^{(t)} = \sum_{i=1}^{m} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{i=1}^{t} \Omega(f_i)$$

Here the $\Omega(f_i)$ represents the regularization term and and $l\left(y_i, \hat{y}_i^{(t)}\right)$ is the loss function. In the Willingness Problem I am choosing logistic loss and in the Duration problem I am choosing softmax multinomial loss.

Besides, we need to tune the max depth of the boosting trees to avoid overfitting. After experiment, I choose max depth of 5 for each subproblem.

V. Experiments, Results and Discussion

A. Experiments

For logistic regression model, decision tree model and XGBoost model, I've tuned the parameters using my spitted dev dataset. In general, my approach to tune parameters are like:

1. Fix other parameters
2. Provide a vector of possible values I want to try
3. Train the model and report on dev AUC performance
4. Pick the optimal one and repeat steps to tune other parameters.

I did not implement cross validation to tune features because my entire dataset is relatively big, and I would rather spend more time on trying different models and resolve two sub problems. I am also aware that I did not cover all permutations of possible choice of multiple parameters and just tuned them one-by-one, which is the same reason of time limitation.

For logistic regression model, I tuned the regularization parameter C and learning rate.

For decision tree, I tuned the minimum sample on leaves.

For XGBoost, I tuned the maximum depth and learning rate.

For neutral network, I investigated and experimented the tradeoff between a reasonable training time v.s. the complexity of model and decided to build a two-hidden-layer neutral network. The number of neurons in each hidden layer is experimented with different combinations. And I've also chosen the epochs as I gradually increase the number until I saw no reasonable improvement. Unfortunately, I could not drive a conclusion on the relationship between the dev result and the network structure based on my current knowledge.

B. Results & Discussion

1) Willingness Problem, we are reporting the accuracy, average precision and ROC AUC score for each model as evaluation metrics. I've also plotted the result ROC curve and Precision-Recall curve for each algorithm to build up an intuition on the model performance. Here I am also listing the training accuracy as a reference.

TABLE I. RESULT FOR WILLINGNESS PROBLEM

Models	Metrics			
	Training Accuracy	***Test Accuracy***	***Test Average Precision***	***Test ROC AUC Score***
Logistic Regression	0.7102	0.7120	0.7482	0.7730
Neutral Network	0.7297	0.7324	0.7810	0.8018
Simple Decision Tree	0.7494	0.7258	0.7758	0.7968
XGBoost	**0.7429**	**0.7367**	**0.7859**	**0.8165**

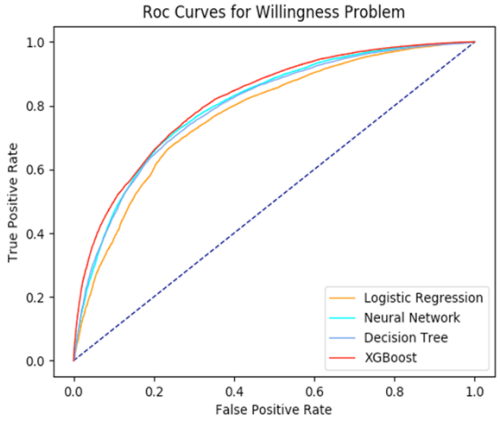


Fig. 1. ROC Curves for Willingness Problem

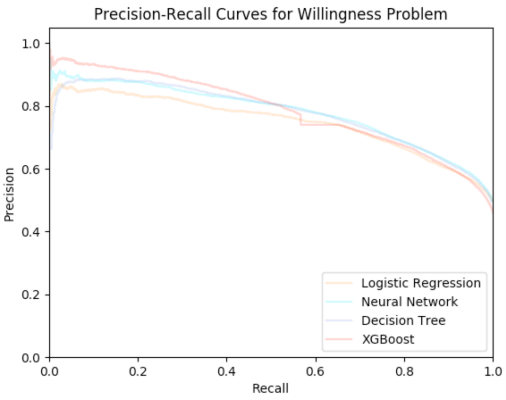


Fig. 2. Precision-Recall Curves for Willingness Problem

As we can tell from the figures and result table, it turns out that XGBoost performs best overall, while other algorithms also giving out reasonably well results. Here is some discussion I derived from results:

- The features I selected are able to reasonably formulate the likelihood of people's willingness to provide us commute preference
- XGBoost is again proved to perform pretty well in structured data and successfully avoided underfitting and overfitting without too much tuning on parameters.
- I kept the missing values in dataset before I feed it to XGBoost because the algorithm package can naturally handle missing values. I am imputing missing values for all other three algorithms which may bias the training data. This may be another reason that XGBoost performed best.

2) Duration Problem, This is a multinomial classfication problem. I've reported the accuracy, precisions, recalls and F1-scores for each model and each label, listing in the below table:

TABLE II. RESULT FOR DURATION PROBLEM

Models	Metrics				
	Training Accuracy	***Test Accuracy***	***Test Precisions***	***Test Recalls***	***Test F1-Scores***
Logistic Regression	0.4766	0.4758	0 – 0.5694 1 – 0.6335 2 – 0.1085	0 – 0.6228 1 – 0.3278 2 – 0.3774	0 – 0.5950 1 – 0.4320 2 – 0.1686
Neutral Network	0.5014	0.4954	0 – 0.5711 1 – 0.6142 2 – 0.1149	0 – 0.6702 1 – 0.3281 2 – 0.3205	0 – 0.6167 1 – 0.4278 2 – 0.1692
Simple Decision Tree	0.4779	0.4539	0 – 0.5955 1 – 0.6003 2 – 0.1082	0 – 0.5893 1 – 0.3015 2 – 0.4691	0 – 0.5923 1 – 0.4014 2 – 0.1758
XGBoost	0.6023	0.5845	0 – 0.5727 1 – 0.6115 2 – 0.2500	0 – 0.8084 1 – 0.4224 2 – 0.0032	0 – 0.6704 1 – 0.4997 2 – 0.0063

With the limitation of page, I will only plot the ROC curve for XGBoost:

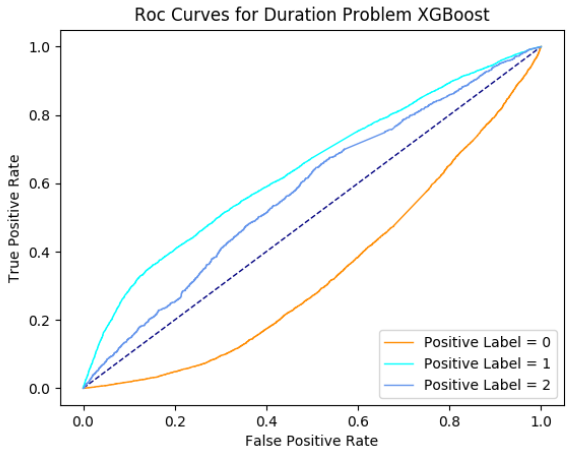


Fig. 3. ROC Curves for Duration Problem (XGBoost)

From the result here, we can note that all models are not performing very well in duration problem. XGBoost seems to perform best but it actually performs poorly on certain category and thus making it not as useful as the willingness problem. Here are some thoughts from me about the possible reasons of the performance:

- The amount of data in label 2 (90, 120) is too low, even though we deal with imbalanced data with weighted loss, it is likely that the data itself is not rich enough to provide signal to classify this category.
- The default setting of our commute preference collection feature is 30mins. This could bias the source data as a lot of people just did not update this time, even though they may really want to commute in other duration of time
- The quality of the latitude / longitude data may not be precise enough to predict in this level of granularity, but good enough to predict if they are commute sensitive.

VI. Conclusion & Future Work

In conclusion, we got good result to predict if a LinkedIn job seeker is willing to provide us information about their commute preference with XGBoost algorithm. We should be able to leverage this result for feature promption as well as using it as another feature in job recommendation. However, we did not result a reasonable result to predict member's potential commute duration time based on the features I produced.

As for the future work with enough time provided, I see opportunities in multiple ways to further improve the model:

1) Better Missing Value Handling. Given that there are some correlations between the feature columns, e.g. job view and job apply, we could potentially implement some regression algorithm to impute the missing values, which takes the populated fields as feature and populate the missing fields.

2) Leverage RNN (Recurrent Neutral Network). I am not "flattening" the job view / apply / save distance features by calculating metrics like count, average and standardized deviation. With my research, I found that RNN is a model to handle vectorized features in a more natral way and I expect it providing better performance on model.

References

[1] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *Journal of machine learning research* 12.Oct (2011): 2825-2830.

[2] Chollet, François. "Keras: Deep learning library for theano and tensorflow." *URL: https://keras. io/k* 7.8 (2015).

[3] McKinney, W. "Python Data Analysis Library—pandas: Python Data Analysis Library [WWW Document]." URL http://pandas. pydata. org/(accessed 7.22. 15) (2015).

[4] Veness, Chris. "Calculate distance and bearing between two Latitude/Longitude points using Haversine formula in JavaScript." Movable Type Scripts (2011).R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[5] Ng, Andrew. "CS229 Lecture notes." CS229 Lecture notes 1.1 (2000): 1-3.

[6] Brownlee, Jason. "8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset." *URL: https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/*

[7] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016.

Project Code

I cannot share the project code as it contains private information about our dataset. But here is a description of my coding work in this project: (1) Significant amount of offline Pig Script to retrieve our data from Hadoop FS. (2) Leveraged Python's pandas to deal with raw data output from hdfs and data preprocess. (3) Leveraged Python's sklearn library and keras library for learning algorithms.