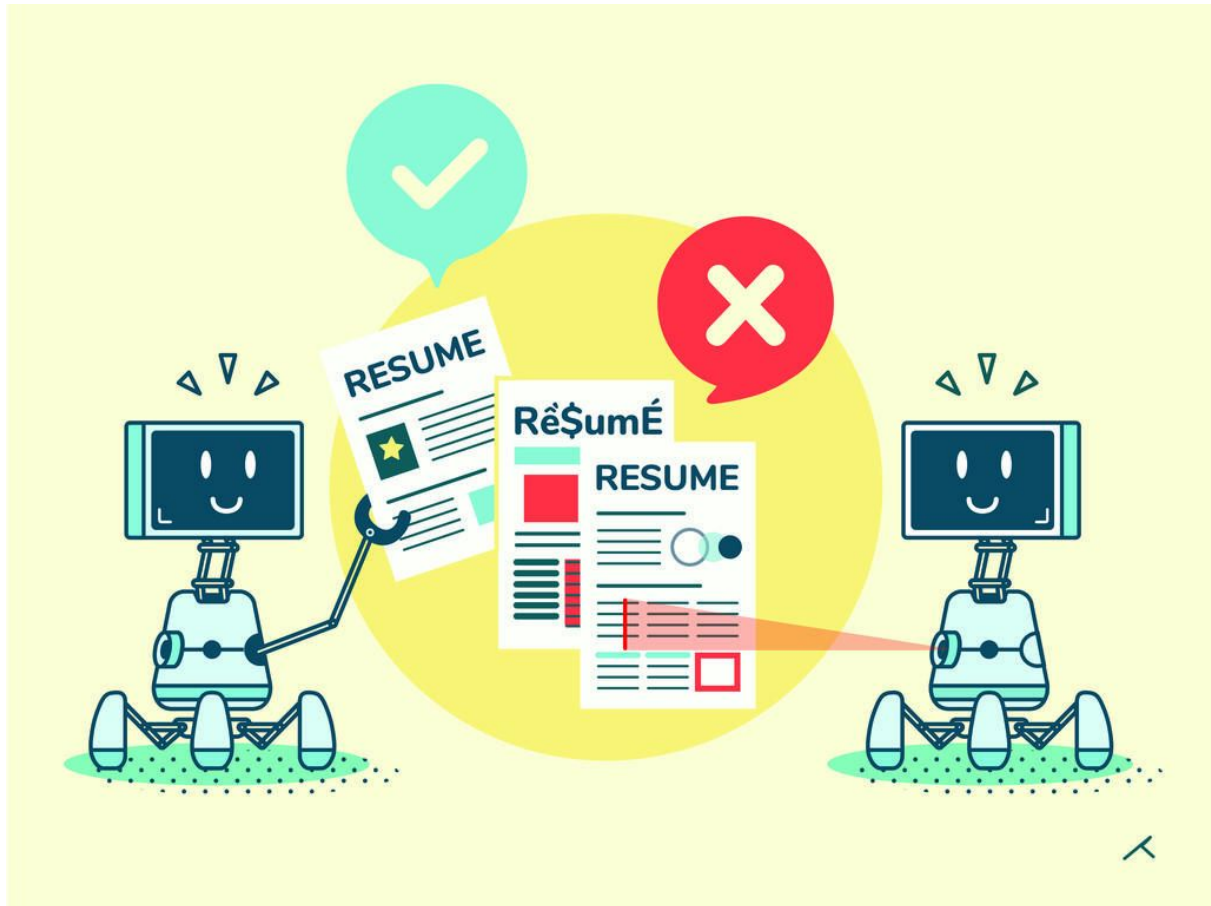


Intelligence Reasoning Systems, Reasoning System and Machine Reasoning

IRS-MRS-F2M2HRSsystem



Authors: team **F2M2**

Ng Siew Pheng (A0198525R)

Tarun Rajkumar (A0198522X)

Tea Lee Seng (A0198538J)

Yang Xiaoyan (A0056720L)

| | |
|--|-----------|
| Executive Summary | 3 |
| Objective | 3 |
| Solution | 4 |
| Overview | 4 |
| Flow Diagram | 4 |
| Searches | 5 |
| Text Normalization | 5 |
| Term Frequency-Inverse Document Frequency (tf-idf) | 5 |
| K-Nearest neighbors | 5 |
| Cosine Similarity | 8 |
| Classification | 12 |
| One-vs-the-rest (OvR) multiclass/multilabel strategy | 14 |
| Create Resume | 16 |
| Optimization (Optaplanner) | 18 |
| Conclusion | 21 |

1. Executive Summary

HR System comprised of many different modules and functions. Our project will be focusing on Recruitment management module and Scheduling & rota management module of an HR system.

After posting a job position to various job agencies, hiring manager will start receiving resumes from HR to review and select candidates. What if there are more than one position to fill in? Having to review every line in every resume to identify the potential candidates, soon becomes a chore and many valuable time is taken up to go through these resumes. One of our objectives is to help managers deal with this issue, by filtering through these resumes and find and recommend the best set of candidates.

At the same time, when HR manager upload the resume into the system, they can also check what other positions might be suitable for the submitted resume. Potentially forward it to other hiring manager for their review.

And what will happen after the staff(s) are hired? How should the manager, organize and plan out the team structure to ensure skills are matching and team budget is met. The scheduling & rota management module is set up to help manager to manage this.

2. Objective

For the unstructured data, extracted from resumes and job descriptions, we need to perform text pre-processing, for example, text normalization and vectorization, to extract features of the document. These features are fitted into the ML algorithms to predict and find the best match resumes to job descriptions.

In the Search Resume function, user will input the important / mandatory search keywords as well as optional search keywords. These are loaded in a sparse matrix and the matrix is loaded into the fitted model and to generate the score.

In the Classify Resume function, user can select a resume and upload it to check job positions that might be suitable for the resume. This will allow the HR team to potentially identify applicants for other positions in the organization.

With new hires, it is important to ensure that teams are formed with matching skills and within budget. Using optaPlanner, we created an optimizer, that resolve constraints from the required skill set and team budget, to generate a recommendation on how the teams should be formed.

3. Solution

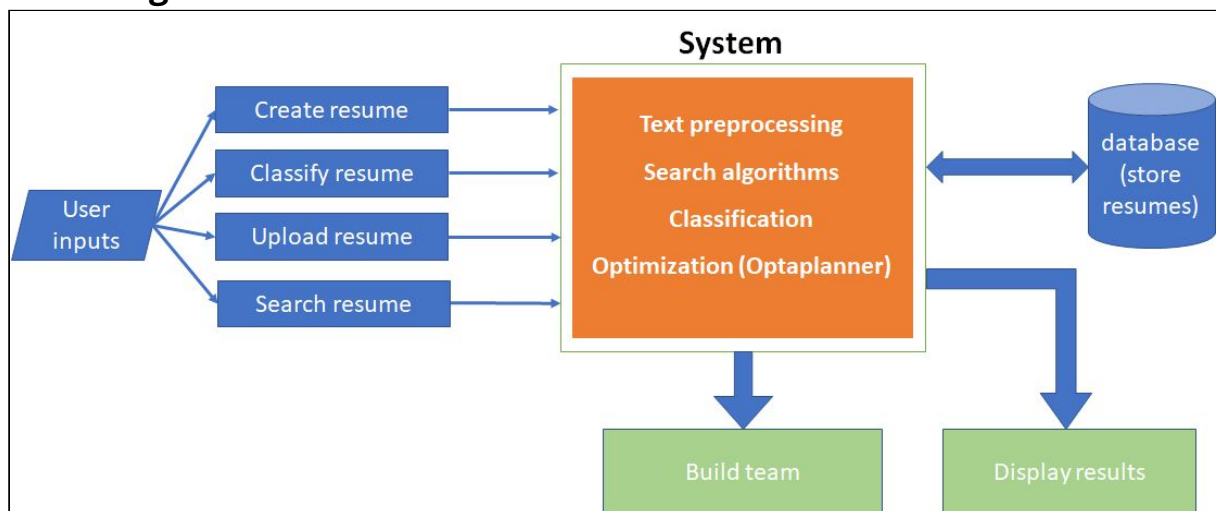
Build an HR System apply machine learning algorithms, input the search keys to the system, it will help the hiring manager to sort out the suitable candidates. The HR system can find a suitable job for a resume. The system also can build a best team based on the skillsets score and team budget.

3.1. Overview

The HR system consists of parts include:

- UI for user inputs and display results
- Search algorithms to match the job requirement and resumes
- Classifier to find a suitable job for a resume
- Optimization to build a team with the rules
- Database to store resumes

Flow Diagram



3.2. Search Resume

3.2.1. Text Normalization

Before text processing, we need to use text normalization techniques to remove unnecessary string or update the text for both search keys and the text in resumes..

- Update the text to lowercase. Eg. update "Manager" to "manager".
- Replace numbers occurrences in string with textual representation. Eg. replace "100" to "one hundred".
- Remove punctuation in the text. Eg. Remove ",", "?" "." etc.
- Remove white spaces after split the searching words into single words. Eg. Update "degree developer" to "degree" "developer"
- Remove stopwords. Eg. remove "the" "that" "and"
- Remove non-ascii characters.
- Lemmatize the text into the base form. Eg. "worked" to "work", "developed" to "develop"

3.2.2. Term Frequency-Inverse Document Frequency (tf-idf)

After normalizing the text, we passed them to tf-idf to rank which document is most relevant to the query, we created a tf-idf vectorizer, fit and transform the search keys or the resume text. The vectors consists of the weight of each word which will be used for the next step for the comparison.

Below is an example code of the search key tf-idf vectorizer:

```
vectorizer = TfidfVectorizer(stop_words='english')
vectorizer.fit(textimp)
vector = vectorizer.transform(textimp)
Job_Desc_Imp = vector.toarray()
```

3.2.3. K-Nearest neighbors

One of the classification algorithms we used in the HR system is k-nearest neighbors (KNN). There are two types of search keys for the algorithm: important key and optional key.

The number of neighbors is one, we used this algorithm to find the distances between the search keys and each resume. For the overall distance, important key contribute 70% and optional key contribute 30%. After getting all the distances we ranking the resumes by the distance to find some suitable candidates for the jobs.

Below is the python code for the distance (score) and ranking for the resumes:

```
for i in resume_vect:
    samples = i
    neigh = NearestNeighbors(n_neighbors=1)
    neigh.fit(samples)
    NearestNeighbors(algorithm='auto', leaf_size=30)
    scorea = neigh.kneighbors(Job_Desc_Imp)[0][0].tolist()
    score_A.append(scorea[0])
    if len(optionalkey) != 0:
        scoreb = neigh.kneighbors(Job_Desc_Opt)[0][0].tolist()
        score_B.append(scoreb[0])

df['Score_A'] = score_A

if len(optionalkey) != 0:
    df['Score_B'] = score_B
    df['Score'] = df['Score_A'] * 0.7 + df['Score_B'] * 0.3
else:
    df['Score'] = df['Score_A']

df = df.sort_values(by=["Score"])
df1 = df[['name', 'profileURL', 'Score']]
print(df1)
flask_return = []

rank = 0
global hasA
global hasB
for idx, row in df1.head(20).iterrows():
    name = row['name']
    filename = row['profileURL']
    score = row['Score']
    if score < 1:
        type = 'typeA'
        hasA = 'hasA'
    elif (score >= 1 and score < 2):
        type = 'typeB'
        hasB = 'hasB'
    else:
        type = 'noType'
    rank = rank + 1
    res = ResultElement(rank, name, filename, score, type)
    flask_return.append(res)
return flask_return
```

We type search keywords into “Mandatory Search Key” and “Optional Search Key” field and select “KNN” option.


Search Resume Page

Mandatory Search Key

Optional Search Key

☒ KNN ☐ Cosine Similarity

[Search](#)



Results about “machine learning data python reinforcement google” and “computer vision” search.

Search Resume Page Result


Best Match Resumes

| Rank | Name | Resume | Score |
|------|----------------------|---|-------|
| 1 | Faizal Ahmad | https://www.linkedin.com/in/faizal-ahmad-90202142/ | 0.524 |
| 2 | Jingwen Zhou | https://www.linkedin.com/in/jingwen-zhou-03926435/ | 0.700 |
| 3 | Mo Yang | https://www.linkedin.com/in/mo-yang-7b20b080/ | 0.700 |
| 4 | Park Kim Hee | https://www.linkedin.com/in/park-kim-hee-108b85137/ | 0.700 |
| 5 | Ryan Law | https://www.linkedin.com/in/ryanlawrlhh/ | 0.700 |
| 6 | Peter | PWC_Olivia_Peter_Regulatory_Manager.pdf | 0.700 |
| 7 | WongZhongMing | PWC_Wong_Zhong_Ming_Senior_Audit_Associate.pdf | 0.700 |
| 8 | Zhenwei Zhao | https://www.linkedin.com/in/zhenwei-zhao-36887123/ | 0.704 |
| 9 | Geraldine Tay | https://www.linkedin.com/in/geraldinetayhuiling/ | 0.724 |
| 10 | Dang-Khoa Le Tan | https://www.linkedin.com/in/letandangkhoea/ | 0.749 |
| 11 | Ming Gao | https://www.linkedin.com/in/ming-gao-57509a101/ | 0.761 |
| 12 | Max Pan | https://www.linkedin.com/in/maxpan/ | 0.782 |
| 13 | Hardian Lawi | https://www.linkedin.com/in/hardianlawi/ | 0.783 |
| 14 | Celestine Lau | https://www.linkedin.com/in/laucelestine/ | 0.836 |
| 15 | Penny | PWC_Penny_Lim_Risk_AM.pdf | 0.836 |
| 16 | Lanani | Luminance_Fund_Saleem_Lalani_Research_Associate.pdf | 0.836 |
| 17 | Yulius Mulyadi Halim | https://www.linkedin.com/in/yuliusmh/ | 0.836 |
| 18 | Sean Ang | https://www.linkedin.com/in/sean-ang-hd/ | 0.836 |
| 19 | Ivy | Lion_Global_Ivy_Choo_Investment_Compliance_Assistant_Manager.docx | 0.836 |
| 20 | Norman Lim | https://www.linkedin.com/in/norman-lim-600a83118/ | 0.836 |

Recommended Resumes

| Rank | Name | Resume Score |
|------|------|--------------|
|------|------|--------------|

[Back to Menu](#)



3.2.4. Cosine Similarity

One more algorithm we used in the HR system is Cosine Similarity which is quite similar algorithm to the KNN. Cosine Similarity measures the similarity between two tf-idf sparse matrix vectors (search key and resume) by measuring the cosine of the angle between them.

Below is the python code about calculating the score and ranks the resume by using Cosine Similarity:

```
vals = cosine_similarity(search_sm, resume_sm)
df = resume_df
df['Score'] = vals[0]

if max(df['Score']) != 0:
    df['NScore'] = df['Score'] / max(df['Score']) #rescale for optaPlannerplanning
else:
    df['NScore'] = df['Score']

df = df.sort_values(by=["Score"], ascending=False)
db = lk_parser.loadData(app_constants.RESUMEDB_FILE_PB)
df['expectedMonthlySalary'] = df['profileURL'].apply(lambda x:
getExpectMonthlySalary(db, x))

df1 = df[['name', 'profileURL', 'Score', 'NScore', 'expectedMonthlySalary']]
return df1
```

Search Resume Page

Mandatory Search Key


machine learning data python reinforcement google

Optional Search Key

☐ KNN

☒ Cosine Similarity

Search



We type search keywords into “Mandatory Search Key” field and select “Cosine Similarity” option.

Search Resume Page Result


Best Match Resumes

| Rank | Name | Resume | Score |
|------|------------|---|-------|
| 1 | Ming Gao | https://www.linkedin.com/in/ming-gao-57509a101/ | 1.000 |
| 2 | Norman Lim | https://www.linkedin.com/in/norman-lim-600a83118/ | 0.754 |

Recommended Resumes

| Rank | Name | Resume | Score |
|------|------------------|---|-------|
| 3 | Max Pan | https://www.linkedin.com/in/maxpan/ | 0.689 |
| 4 | Hardian Lawi | https://www.linkedin.com/in/hardianlawi/ | 0.626 |
| 5 | Jack Sim | https://www.linkedin.com/in/jack-sim-14887b33/ | 0.595 |
| 6 | Dang-Khoa Le Tan | https://www.linkedin.com/in/letandangkhoea/ | 0.568 |
| 7 | Wen Fong Koh | https://www.linkedin.com/in/koh-wenfong/ | 0.445 |
| 8 | Zhenwei Zhao | https://www.linkedin.com/in/zhenwei-zhao-36887123/ | 0.443 |
| 9 | Haoming Guo | https://www.linkedin.com/in/haoming-guo-16854a173/ | 0.433 |
| 10 | Faizal Ahmad | https://www.linkedin.com/in/faizal-ahmad-90202142/ | 0.425 |

[Back to Menu](#)



Results about “machine learning data python reinforcement google” search.

Search Resume Page Result


Best Match Resumes

| Rank | Name | Resume | Score |
|------|--------------|---|-------|
| 1 | Kaiting Yang | https://www.linkedin.com/in/kaiting-yang/ | 0.102 |
| 2 | Gloria | CV-Gloria_Cheng2018.docx | 0.081 |
| 3 | Vasanti | 180517_Vasanthi_Kasinathan.docx | 0.070 |
| 4 | Penny | PWC_Penny_Lim_Risk_AM.pdf | 0.048 |
| 5 | Lanani | Luminance_Fund_Saleem_Lalani_Research_Associate.pdf | 0.032 |
| 6 | Jingwen Zhou | https://www.linkedin.com/in/jingwen-zhou-03926435/ | 0.023 |
| 7 | Max Pan | https://www.linkedin.com/in/maxpan/ | 0.006 |

Recommended Resumes

Not found recommended resumes

[Back to Menu](#)



We find the search result is useful as different search keywords returns us vastly different profiles. Above is the result of search about “finance”

3.3. Resume Classification

As part of the application initialisation, our classifier model is trained and modeled with the Dataset.

```
class resumeClassifier():
    def __init__(self):
        self.model, self.label_encoder, self.vectoriser = get_trainedModel()

def get_trainedModel():
    resumeDataSet = pd.read_csv('UpdatedResumeDataSet.csv', encoding='utf-8')
    resumeDataSet['cleaned_resume'] = ''
    resumeDataSet['cleaned_resume'] = resumeDataSet.Resume.apply(lambda x: cleanResume(x))

    var_mod = ['Category']
    le = LabelEncoder()
    for i in var_mod:
        resumeDataSet[i] = le.fit_transform(resumeDataSet[i])

    requiredText = resumeDataSet['cleaned_resume'].values
    requiredTarget = resumeDataSet['Category'].values

    word_vectorizer = TfidfVectorizer(
        sublinear_tf=True,
        stop_words='english',
        max_features=1500)
    word_vectorizer.fit(requiredText)
    WordFeatures = word_vectorizer.transform(requiredText)

    clf = OneVsRestClassifier(KNeighborsClassifier())
    clf.fit(WordFeatures, requiredTarget)
    return clf, le, word_vectorizer
```

Post which a model to extract features (word_vectorizer) a label_idenfier (le) and the classifier model (clf) are used to create the features for a given resume, label to match against and the job title/position classifier are globally used in the application.

Choose one or more files

Browse

Submit

We input one or more resume/cv which are stored in our resume Database.

Each resume is then pre-processed to extract plaintext from the pdf file provided

```
def extract_text_from_pdf(pdf_file):
    pdfreader = PyPDF2.PdfFileReader(open(pdf_file, 'rb'))
    pdf_content = ""
    for page in range(0, pdfreader.getNumPages()):
        pdfpage = pdfreader.getPage(page)
        pdf_content = pdf_content + pdfpage.extractText()
    return cleanResume(pdf_content)

def cleanResume(resumeText):
    resumeText = re.sub('http\S+\S*', ' ', resumeText)
    resumeText = re.sub('RT|cc', ' ', resumeText)
    resumeText = re.sub('#\S+', ' ', resumeText)
    resumeText = re.sub('@\S+', ' ', resumeText)
    resumeText = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~"""), ' ', resumeText)
    resumeText = re.sub(r'[\x00-\x7f]', r' ', resumeText)
    resumeText = re.sub('\s+', ' ', resumeText) # remove extra whitespace
    return resumeText
```

The classification is done as follows

```
@app.route('/resumeSubmit', methods=['GET', 'POST'])
def get_resume():
    # data = json.dumps(request.form)
    print(request.files['resumes'])
    resumes = []
    for name, file in request.files.items():
        filename = uploadFile(file)
        resume = classifier.extract_text_from_pdf(app.config['UPLOAD_FOLDER']+filename)
        info = classifierModel.label_encoder.classes_[classifierModel.model.predict(
            classifierModel.vectoriser.transform([resume]))]
        resumes.append((filename, info))

    return render_template('classifierResult.html', results=resumes)
```

3.3.1. One-vs-the-rest (OvR) multiclass/multilabel strategy

The underlying principle to match resumes to its associated job title/position is OvR classifier. Since we did not have a large dataset, we used OvR. In this model when a particular label needs to be matched all the other labels are considered to be negative. Another parameter that was considered here was the usage of 'KNeighborsClassifier' as the estimator. This combination yields an accuracy of 99%.

```

: clf = OneVsRestClassifier(KNeighborsClassifier())
  clf.fit(X_train, y_train)
  prediction = clf.predict(X_test)
  print('Accuracy of KNeighbors Classifier on training set: {:.2f}'.format(clf.score(X_train, y_train)))
  print('Accuracy of KNeighbors Classifier on test set: {:.2f}'.format(clf.score(X_test, y_test)))

  print("\n Classification report for classifier %s:\n%s\n" % (clf, metrics.classification_report(y_test, prediction)))

```

Accuracy of KNeighbors Classifier on training set: 0.99
Accuracy of KNeighbors Classifier on test set: 0.99

Classification report for classifier OneVsRestClassifier(estimator=KNeighborsClassifier(algorithm='auto',
leaf_size=30,
metric='minkowski',
metric_params=None,
n_jobs=None, n_neighbors=5,
p=2, weights='uniform'),
n_jobs=None):

Expected result is presented as a table

| # | Resume | Suggested Role |
|---|-------------------|------------------|
| 1 | TarunRajkumar.pdf | ['Data Science'] |

[Back to Menu](#)

3.4. Optimization (Optaplanner)

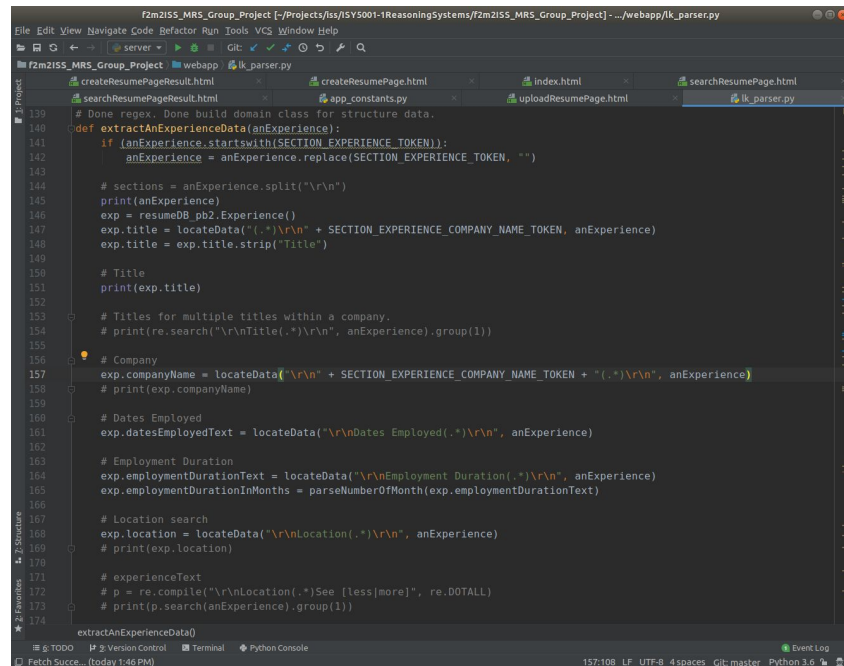
3.4.1. Create Resume

As part of the staff onboarding process, HR would need to upload the staff information, for example their resume, their position and also monthly salary. We created this function to mimic that process.

The image shows a web form titled "Create Resume Page" overlaid on a background image of a person standing on a grassy hill. The form contains the following sections and input fields:

- Name**: A single-line text input field.
- Linkedin Profile URL**: A single-line text input field containing the placeholder text "https://.....".
- Role**: A single-line text input field containing the placeholder text "at".
- Monthly Salary**: A single-line text input field containing the placeholder text "0.0".
- About**: A large multi-line text area.
- Experience**: A large multi-line text area.
- Education**: A large multi-line text area.
- Licenses & Certifications**: A large multi-line text area.

For “Experience” and “Education” sections, the app parsing info into detailed fields.



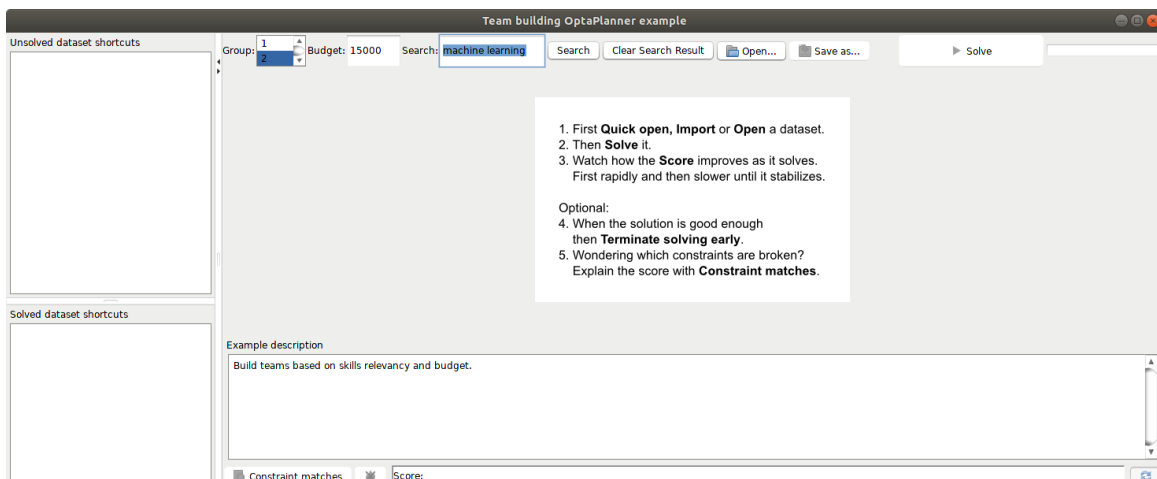
```
139 # Done regex. Done build domain class for structure data.
140 def extractAnExperienceData(anExperience):
141     if (anExperience.startswith(SECTION_EXPERIENCE_TOKEN)):
142         anExperience = anExperience.replace(SECTION_EXPERIENCE_TOKEN, "")
143
144     # sections = anExperience.split("\r\n")
145     print(anExperience)
146     exp = resumeDB_pb2.Experience()
147     exp.title = locateData("(.*?)\r\n" + SECTION_EXPERIENCE_COMPANY_NAME_TOKEN, anExperience)
148     exp.title = exp.title.strip("Title")
149
150     # Title
151     print(exp.title)
152
153     # Titles for multiple titles within a company.
154     # print(re.search("\r\nTitle(.*?)\r\n", anExperience).group(1))
155
156     # Company
157     exp.companyName = locateData("(.*?)\r\n" + SECTION_EXPERIENCE_COMPANY_NAME_TOKEN + "(.*?)\r\n", anExperience)
158     # print(exp.companyName)
159
160     # Dates Employed
161     exp.datesEmployedText = locateData("\r\nDates Employed(.*?)\r\n", anExperience)
162
163     # Employment Duration
164     exp.employmentDurationText = locateData("\r\nEmployment Duration(.*?)\r\n", anExperience)
165     exp.employmentDurationInMonths = parseNumberOfMonth(exp.employmentDurationText)
166
167     # Location search
168     exp.location = locateData("\r\nLocation(.*?)\r\n", anExperience)
169     # print(exp.location)
170
171     # experienceText
172     # p = re.compile("\r\nLocation(.*?)See [less|more]", re.DOTALL)
173     # print(p.search(anExperience).group(1))
174
175     extractAnExperienceData()
```

Structure data are stored in ProtoBuf file and potentially usable for optaPlanner planning.

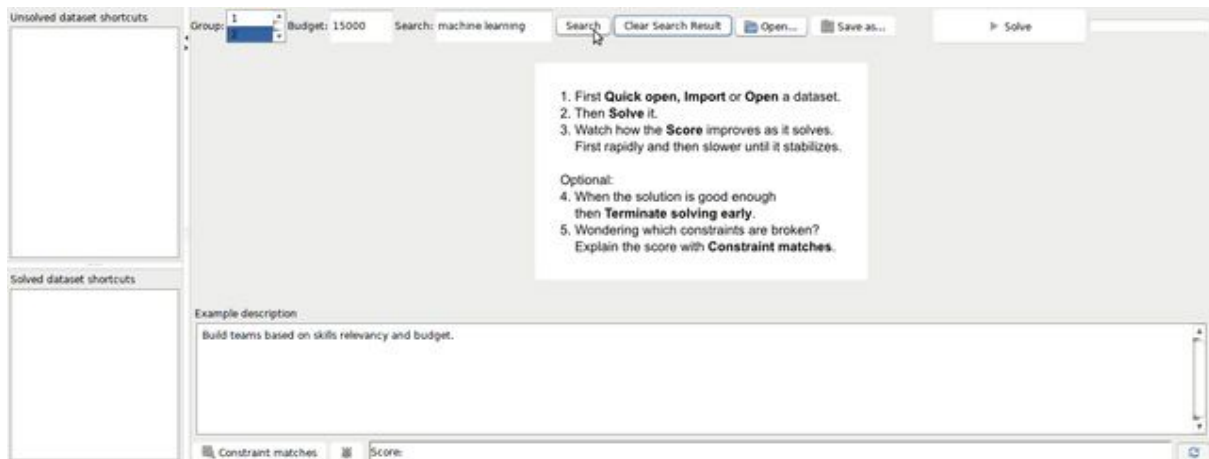
3.4.2. Optimizer (using optaPlanner)

We created optaPlanner to optimize team recruitment based on skill sets relevancy and budget.

First, we can search relevant skill sets of candidates by firing rest API call to http://server.leeseng.tech:5000/api_search . The implementation is the same as search in cosine similarity.



API server returns the list of candidates. We customized excel generator to estimates potential tiers required for planning with respect to number of candidates returned. Without that estimation, our planning always failed to meet budget as too many candidates are being forced to be planned within small groups.



As our candidates result set is small, planning usually stabilized within seconds.

The screenshot shows the HR machine learning tool interface with the results of a search for 'machine learning' with a budget of 15000. The results are displayed in a table with columns for Groups, Tiers, and Candidates. The table is titled 'Team building - HR-machine-learning-14Candidates-4Tiers-2Groups.xlsx'.

| Groups | Tier 0 | Tier 1 | Tier 2 | Tier 3 | Unassigned |
|------------|--|---------------------------------------|-----------------------------------|--------|------------|
| G 1 | Faizal Ahmad 0.35 Salary:4100.0 | Norman Lim 0.23 Salary:4200.0 | Sean Ang 0.05 Salary:5600.0 | | |
| | Ming Gao 0.50 Salary:5700.0 | Geraldine Tay 0.31 Salary:4500.0 | David Won 0.11 Salary:4500.0 | | |
| | Hardian Lawi 0.76 Salary:5200.0 | Celestine Lau 0.15 Salary:5200.0 | | | |
| G 2 | Max Pan 0.61 Salary:6500.0 | Kate (Yu) Huang 0.39 Salary:7800.0 | Vikas Dubey 0.04 Salary:4500.0 | | |
| | Dang-Khoa Le Tan 0.83 Salary:6300.0 | Zhenwei Zhao 0.39 Salary:6400.0 | | | |
| Unassigned | | | | | |

At the bottom of the table, there's a 'Latest best score: 0hard/-653medium/0soft'.

From the results above, groups in Tier 0 is preferred than other tiers.

Though the planning looks complicated during, the core reasoning to move candidates around in groups and tiers, is governed by the following 2 rules.

```

78 then
79     scoreHolder.penalize(kcontext);
80 end
81 */
82
83 rule "Team within budget"
84 when
85     $leftAssignment : MeetingAssignment(room != null, $room : room, $leftId : id)
86     accumulate
87         $assignment : MeetingAssignment(room == $room, calculateOverlap($leftAssignment) > 0);
88         $total : sum($assignment.getMeeting().getSalary());
89     then
90         System.out.println(assignment);
91         scoreHolder.addMediumConstraintMatch(kcontext, $total > $room.getSalaryBudget() ? ($room.getSalaryBudget() - $total) : 0); //1+2, self+ other //addHardConstraint
92 end
93
94
95 rule "Do all meetings as soon as possible"
96 when
97     MeetingAssignment(startingTimeGrain != null, $startingTimeIndex : startingTimeGrain.getGrainIndex(), $meeting : meeting)
98 then
99     scoreHolder.addMediumConstraintMatch(kcontext, (int)-(($startingTimeIndex+1)*($meeting.getNscore()*100));
100 end
101

```

Due to rapid development, we customized optaPlanner example, Meeting Scheduling. We reuse Room class for grouping, TimeGrain class for Tier and Meeting class to represent a candidate.

Constraint matches view also explain penalty incurred for each candidate in current plan.

| Constraint name | Constraint w... | Match count | Score |
|-------------------------------------|-----------------|-------------|------------|
| Do all meetings as soon as possible | 1soft | 13 | -653medium |
| Team within budget | N/A | 13 | 0 |

Constraint matches of selected constraint type

- [David Won 0.11 Salary:4500.0] = -32medium
- [Vikas Dubey 0.04 Salary:4500.0] = -11medium
- [Hardian Lawi 0.76 Salary:5200.0] = -76medium
- [Sean Ang 0.05 Salary:5600.0] = -16medium
- [Zhenwei Zhao 0.39 Salary:6400.0] = -77medium
- [Celestine Lau 0.15 Salary:5200.0] = -30medium
- [Ming Gao 0.50 Salary:5700.0] = -49medium
- [Dang-Khoa Le Tan 0.83 Salary:6300.0] = -82medium
- [Geraldine Tay 0.31 Salary:4500.0] = -61medium
- [Max Pan 0.61 Salary:6500.0] = -60medium
- [Kate (Yu) Huang 0.39 Salary:7800.0] = -78medium
- [Faizal Ahmad 0.35 Salary:4100.0] = -35medium
- [Norman Lim 0.23 Salary:4200.0] = -46medium

We created a heuristic formula for Tier penalty calculation by normalized skillsets relevancy.

$$(int)-((\$startingTimeIndex+1)*(\$meeting.getNscore()*100))$$

3.4.3. Integration Challenges with optaPlanner

With optaPlanner examples, optaPlanner is providing great UIs about the planning results and changes. However, it is still challenging for recent integration.

1. For Meeting scheduling example, they use excel file as storage. That introduced extra data persistence code for new data columns. XML file storage was used previously, and new data columns persists without extra coding.
2. No extra UI container area for custom UI. Good news is that, we still can change common UI code to add in our search field, as it is open source.

4. Conclusion

TD-IDF, KNN and cosine similarity are great algorithms to search relevant resume. They served as a good foundation for text related ML problem.

We can even quantify relevancy into a metric and utilized the metric as planning constraints. This is where optaPlanner comes in. It is very concise in defining constraints, that are critical for core planning logic management.

We were able to apply what we learnt in the IRS module. Taking unstructured data like resumes and job descriptions, we can use model to turn them into a group of structured representation of knowledge and apply machine reasoning to match jobs and resumes.