Project Phase 4

Foo Zhi Yuan, Chew Ming Chan, Yi Ding

Note: Please note that we have also included our Jupyter notebook describing our end to end process of doing this phase.

Creating the Golden Data

During our phase 3, we have a setA.csv (IMDB) that contains 21700 tuples and a setB.csv (rottenTomatoes) that contains 6921 tuples. Then we performed blocking on them and obtained a setC.csv that contains 23331 tuple pairs.

Originally, we attempted to perform matching using the setC.csv that we created in phase 3 of the project.

However, there are several problems:

a. The number of tuples in setA and the number of tuples in setB are greatly uneven. This causes a lot of the tuples in setA cannot be matched to anything.

b. While we are attempting to label the golden data for setC.csv, we found out that there are way too many negative label in compared to positive label, eg: We planned to label 500 of the tuple pairs, and we split 500 of the tuple pairs into 5 sets, each sets containing 100 tuple pairs. However, we immediately find out that in the first set containing 100 tuple pairs that we try to label, there are only 6 positive label. By normal distribution, 6x5=30. We concluded that 30 positive labels in 500 tuples pairs is way too skewed.

Thus, we decided to redo the blocking. However, to solve the problem of the huge differences in the amount of tuples between setA.csv and setB.csv, we have to start this project all over again. Therefore, we recrawled our data for setB.csv (rottenTomatoes), this time having 19183 tuples in it.

Then, we perfomed blocking on setA.csv and the new setB.csv, and obtain a setC.csv that contains 13523 tuple pairs.

Due to the fact that setC.csv has too many attributes right now, to prevent high time complexity when parsing in setC.csv into py_entitymatching and scikitlearn, we select only a few only the key attributes from setC.csv that is important to determine whether two movies are the same, namely: id, title, category, duration , rating, rating count and directors.

We extract 500 tuples pairs randomly from setC.csv using seed==10, and label it manually, creating a new file called sampleA.csv.

We labelled a match as 1, and not match as 0.

Labelling criteria

We labelled based on the movie name, duration and directors of tableA and tableB. This is because having the same movie name doesn't mean they are the same movie. Thus, although we put heavy emphasis on the movie name, we also label based on the duration and directors.

While labelling, sometimes there are missing value for directors and duration, thus, we can't tell whether tableA and tableB are the same movie just by looking at the csv. In scenario like this, we will search for the movie on IMDB website and RottenTomatoes website, then determine if they are matched based on the information we obtained (pictures, year etc).

Filling up missing value

setC.csv has 273 missing value in attribute 'ltable_Category', 1570 missing values in attribute 'ltable_Rating', 1570 missing values in attribute 'ltable_Rating_Count', 4114 missing values in attribute 'ltable_Director'; 566 missing values in attribute 'rtable_Category', 2274 missing values in attribute 'rtable_Duration', 3228 missing values in attribute 'rtable_Rating' and 1613 missing values in attribute 'rtable_Director'.

In addition to that, our sampleA.csv, which is our golden data has 5 missing values for attribute ‘ltable_Category’, 40 missing values for attribute ‘ltable_Rating’, 40 missing values for attribute ‘ltable_Rating_Count, 113 missing values for attribute ‘ltable_director’, 18 missing values for attribute ‘rtable_Category’, 63 missing values for attribute ‘rtable_Duration’, 98 missing values for attribute ‘rtable_Rating’, 45 missing values for attribute ‘rtable_Director’.

It can be seen that filling in these missing values one by one by searching the internet seems impractical and inefficient. Thus, for all of the rows that have attributes with missing values, we simply fill in 0 for the value.

Creating feature vectors

In setC.csv and sampleA.csv, we have attributes id, title, category, duration, rating, rating count and directors.

To create feature vectors in order determine whether two tuple pairs match:

a. We take in two titles x and y and compute the edit distance between x and y.

```
def title_match(x, y):
        return lev.get_raw_score(x, y)
```

b. We take in attribute “Category” x and y then compute the edit distance between x and y.

```
def category_match(x, y):
        return lev.get_raw_score(x, y)
```

c. We take in the attribute “rating” x and y, then compute the modulus of x minus y.

```
def rating_match(x, y):
        return abs(float(x) - float(y))
```

d. We take in the attribute “director” x and y, then compute the edit distance between x and y.

```
def director_match(x, y):
        return lev.get_raw_score(x, y)
```

e. We take in the attribute “rating_count” x and y, then compute the modulus of x-y.

```
def rating_count_match(x, y):
        return abs(float(x) – float(y))
```

f. We take in the attribute “duration” x and y, then compute the modulus of x-y.

```
def duration_match(x, y):
        return abs(float(x) – float(y))
```

Selecting for the Best matcher

Iteration 1 (Cross Validation)

Recall that in our golden data, sampleA.csv, we have 500 tuples that are being manually labelled. Note that from now on we will refer to sampleA.csv as labelled.csv for the purpose of clarity.

We split labelled.csv into development set and evaluation set, with development set having 375 tuples (from 0 to 374) and the evaluation set having 125 tuples (from 375 to 500).

Using the features a, b, c, d, e, f above, we converted our development set into feature vectors. Then, we performed 4 fold cross validation on it for each of the learning methods: Decision Tree,

Random Forest, Logistic Regression, Support Vector Machine (SVM) and Naive Bayes. We sum up the precision, recall and F of all of the 4 folds and take the average.

However, we also noticed that some of the classifiers like support vector machine and naive bayes have trouble running on our feature vector. Thus, we will temporarily suppress 2 of our feature, eg: duration and rating_count for now.

Num.	**Classifier**	**Precision**	**Recall**	**F1**
1	Decision Tree	0.9678571428571429	0.9928571428571429	0.9796992481203008
2	Logistic Regression	0.9305184773204795	0.901147382029735	0.9137459926080209
3	Random Forest	0.9821428571428572	0.9592766166295579	0.9702627295977679
4	Support Vector Machine	1.0	0.8216931216931217	0.8914576802507836
5	Naive Bayes	0.9004294093003771	0.9082902391725921	0.9027294858564209

Although support vector machine has a precision of 1.0, it is important to note that it has the lowest recall among all other classifiers.

In addition to that, based on our knowledge about support vector machine, the advantages of support vector machines are: Effective in high dimensional spaces. Still effective in cases where number of dimensions is greater than the number of samples. Besides that, SVM uses a subset of training points in the decision function (called support vectors), so it is also memory efficient. Other than that, SVM is also known to be versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels. However, with that being said, the disadvantages of support vector machines include: If the number of

features is much greater than the number of samples, the method is likely to give poor performances. SVMs also do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation

Based on this table (based on F1), we observed that the top 2 most ideal classifier are decision tree and random forest, which we will test this hypothesis later. (Our top classifier is decision tree.)

Debugging the classifier

Most classfiers return an F1-Score of 0.9 and above, the accuracy is consider high. When tested on the isolated evaluation set, the F1-Score is around 0.9 too. But, we will try to make the classifier even better, that is, achieving 1.0 in precision and 1.0 in recall, the task that all data scientists want to achieve.

Some things that we plan to look into are:

1) Conversion of data into feature vectors (Recall that we suppress duration and rating_count, which we will fix now, eg: In iteration 3).

2) Classifier that is used, we might be able to tweak the scikit classifier.

Iteration 2 (Debugging)

We further split our development set earlier into training set and test set in the ratio of 50:50 (Recall that earlier we use our development set to perform 4 fold cross validation).

Differing from the project requirement, instead of only training 1 classifier on training set and test it on test set, we train all 5 of our classifier on the training set and test them on the test set and compute the precision and recall of it. In addition to that, instead of debugging on the classifier which perform the best by looking at the false positive and false negative, we debug on all 5 of the classifier which hopefully can improve our overall precision and recall.

Num.	Classifier	Precision	Recall	F1
1.	Decision Tree	0.8923076923076924	0.9354838709677419	0.9133858267716536
2.	Logistic Regression	0.9344262295081968	0.9193548387096774	0.9268292682926829
3.	Random Forest	0.9508196721311475	0.9354838709677419	0.943089430894309
4.	Support Vector Machine	1.0	0.5967741935483871	0.7474747474747475
5.	Naive Bayes	0.8923076923076924	0.9354838709677419	0.9133858267716536

Below are some of the false negative and false positive examples

1.

ltable: ['16751', 'Death Al Dente: A Gourmet Detective Mystery', 'Crime,Drama,Mystery', '0', '67', '63', 'Terry Ingram']

rtable: ['18707', 'Death al Dente: A Gourmet Detective Mystery', 'Mystery', '120', 0, '0', 'Terry Ingram,']

Label: 1 Predicted: 0

Problem: ltable has “Al” in the middle, rtable has “al” in the middle. We will need to change all of our movie name into lowercase before computing the feature vector.

2.

ltable: ['21448', 'Megamind', 'Animation,Television', '0', 0, 0, '', '2010']

rtable: ['4077', 'Megamind', 'Action,Adventure,Animation,Comedy,Kids,Family', '96', '76', '207608', 'Tom McGrath,', '2010']

Label: 1 Predicted: 0

Problem: In this case, the ltable has rating and rating count of 0 respectively. Due to the way of we computing the feature vector for rating and rating count, eg: use modulus of rating1 minus rating2, in cases like this, the feature vector method will return a very large value (|0-76|=76 (rating) and |0-207608|=207608).

3.

ltable: ['727', 'Underdog', 'Animation,Family,Comedy', '0', '75', '1253', '']

rtable: ['2833', 'Underdog', 'Action,Adventure,Comedy,Kids,Family', '80', '60', '122557', 'Frederik Du Chau,']

Label: 1 Predicted: 0

Problem:

1. In case like this where many attributes value are missing, we need more information in order to tell if two movies are the same. Since we fill in missing values by inserting the value 0, instead of filling in one by one by searching the internet, we decided to add one more attribute that we think is crucial to tell whether 2 movies are the same, eg: year.
2. rtable's "Comedy and Family" are a subset of ltable's "Animation, family, Comedy". However, due to the reason that we implement our feature for Categories using edit distance, it doesn't work well reflecting this fact.

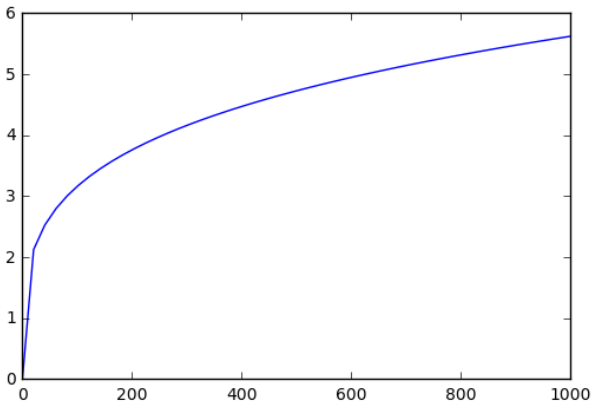
Solutions:

1. Change the movie title feature by converting the 2 movie titles x and y into lower cases before computing the edit distance of them.

2. Initially, we compute the feature of ratings and ratings_count by computing the modulus of their differences, eg: mod(x-y). We decided to square root the modulus of their differences instead, to lower down the differences when one movie has ratings of 0 and another has ratings of a large number.

The reasoning behind using a square root:

```
In [32]: x = lins(0,1000)
         y = list(map(lambda x:math.sqrt(math.sqrt(x)), x))
         plot.plot(x,y)
         plot.show()
```



Although we created the feature of ratings_count, we decided to suppress it because we concluded that the number of people who rated on the movie cannot determine whether two movies are matched.

3. We decided to add in the attribute “year” into our setC.csv to increase precision and recall.

After adding the attribute “year” into our setC.csv, naturally we also need to create a feature for year:

Below is the implementation of our feature for year.

```
def year_match2(x, y):
    if x == y:
        return 1
    return 0
```

4. We decided to modify our feature for duration. Making it such that if 2 durations are the same, return 1. Return 0 otherwise.

5. We will tokenised category by comma. For every token in ltable, if there is a same token in rtable, we will add the count by 1. After that, return the count.

The implementation is as follows.

```
def category_match2(x, y):
    cat_x = x.split(',')
    cat_y = y.split(',')

    count = 0
    for i in cat_x:
        for j in cat_y:
            if i == j:
                count += 1
                return 1
    return 0
```

6. We also tokenised directors by comma. For every token in ltable, we will perform similarity distance with rtable. We set the threshold to be 0.9. If the similarity measure >0.9, return 1. Return 0 if otherwise. The implementation is as follows.

```
def director_match2(x, y):
    dir_x = x.split(',')
    dir_y = x.split(',')

    count = 0
    for i in dir_x:
        for j in dir_y:
            ii = i.strip().lower()
            jj = j.strip().lower()
            if ii and jj:
                if lev.get_sim_score(ii,jj) > 0.9:
                    count += 1
                    return 1
```

Iteration 3 (Cross Validation)

After performing all the solutions suggested above, our feature vectors have changed drastically. Thus, we decided to perform a 4 fold cross validation one more time to select the best classifier again.

Num.	Classifier	Precision	Recall	F1
1.	Decision Tree	0.9535714285714285	0.971218487394958	0.9616821905510043
2.	Logistic Regression	0.9343750000000001	0.8892125739184563	0.9092312292970187
3.	Random Forest	0.9827586206896551	0.9783613445378152	0.9799871441113759
4.	Support Vector Machine	0.9724358974358974	0.8583146591970121	0.910302863156827
5.	Naive Bayes	0.9352761243386244	0.9781512605042018	0.9544795599880346

Again, we observed that support vector machine has the lowest recall among all. Looking at F1, we again observed that decision tree and random forest are the top 2 classifiers among all.

Thus, our earlier hypothesis that the top 2 classifier suitable for tasks like this are decision tree and random forest is valid!

Our top classifier in this iteration is random forest.

Iteration 4 (Debugging)

Now, instead of debugging on all 5 of the classifiers, we decided just to debug on the top 2 classifiers, eg: decision tree and random forest.

Below are the precision and recall of each of the classifiers, having train on training set and test on test set (training set and test set are being splitted from development set in the ratio of 50:50).

Num.	**Classifier**	**Precision**	**Recall**	**F1**
1.	Decision Tree	0.9354838709677419	0.9508196721311475	0.943089430894309
2.	Logistic Regression	0.9636363636363636	0.8688524590163934	0.9137931034482758
3.	Random Forest	0.967741935483871	0.9836065573770492	0.975609756097561
4.	Support Vector Machine	1.0	0.8688524590163934	0.9298245614035088
5.	Naive Bayes	0.9090909090909091	0.9836065573770492	0.9448818897637795

Below are some of the false positive and false negative examples.

1.

ltable: ['16751', 'Death Al Dente: A Gourmet Detective Mystery', 'Crime,Drama,Mystery', '0', '67', '63', 'Terry Ingram', '2016']

rtable: ['18707', 'Death al Dente: A Gourmet Detective Mystery', 'Mystery', '120', 0, '0', 'Terry Ingram,', '2016']

Label: 1 Predicted: 0

2.

ltable: ['3703', 'Persuasion', 'Drama,Romance', '93', '76', '10381', 'Adrian Shergold', '2008']

rtable: ['13876', 'Persuasion', 'Drama,Romance', '102', '78', '22801', 'Roger Michell,', '1995']

Label: 0 Predicted: 1

3.

ltable: ['3426', 'Percy Jackson & the Olympians: The Lightning Thief', 'Adventure,Family,Fantasy', '118', '59', '143056', 'Chris Columbus', '2010']

rtable: ['13539', 'Percy Jackson & the Olympians: The Lightning Thief', 'Action,Adventure,Comedy,Drama,Science Fiction,Fantasy', '83', '66', '253745', 'Chris Columbus,', '2010']

Label: 1 Predicted: 0

Problems and solutions

a. By looking at these false positive and false negative examples, we observed that the rating are having too much weightage than they should to the learning algorithms. Thus, we decided to suppress the "rating" feature vector (recalled that we already suppress the "rating_count" feature since earlier iteration).

b. In addition, we decided to modify our feature for category_match. Instead of using edit distance, which doesn't make sense and seems inaccurate, we decided to first convert x and y into lower case, then use Jaccard measure, tokenize by delimiter ",". However, since we already know before hand that the ltable(IMDB) has category called "News" which the rtable(rottenTomatoes) does not have, any movies with category "News" must not be a matched. Thus, we return -1 if category=="News".

The implementation is as follows:

```
def category_match3(x,y):
    x=x.lower()
    y=y.lower()
    if "news" in x and "news" not in y:
        return -1
    else:
        j=delim_tok.tokenize(x)
        k=delim_tok.tokenize(y)
        results=jac.get_raw_score(j,k)
        return results
```

c. Besides that, we modified our feature for directors. Instead of tokenizing it by the symbol comma and then perform similarity distance on it, and return 1 or 0 based on the threshold, we decided to first convert x and y into lower case, then tokenized them by the symbol comma. We will then return the result of Jaccard measure.

d. Other than that, we modified our feature for title. Instead of returning the similarity score, we will convert x and y into lowercase, then tokenise it by empty space. We will then return the result of Jaccard measure. This change is made because we know that Jaccard measure is more accurate than similarity score.

e. We also introduced a new classifier that we think might work well in making predictions for data like this, that is linear support vector classifications, also known as Linear SVC.

Iteration 5 (Cross Validation)

After adding Linear support vector classification as our 6th learning algorithm, and performing the changes as mentioned in iteration 4 (debugging), we performed 4 fold cross validation again. The precision, recall and F1 results are as follows:

Num.	**Classifier**	**Precision**	**Recall**	**F1**
1.	Decision Tree	0.9751157407407407	0.955026455026455	0.964621337755666
2.	Logistic Regression	0.9832417582417583	0.9596327419856832	0.971130441885159
3.	Random Forest	0.9772272272272272	0.9907407407407407	0.9837962962962963
4.	Support Vector Machine	0.9820512820512821	0.9239184562713975	0.9513502221049391
5.	Naive Bayes	0.9655830280830281	0.9833877995642701	0.9742092055351772
6.	Linear Support Vector Classification	0.9835978835978836	0.9688920012449425	0.976022126022126

Based on the table, there seemed to be a change in our top 2 classifier. Although random forest is still the top 2 among our classifiers, the best classifier right now in terms of precision has become linear support vector classification. Since we focus more on precision rather than recall, we declare linear support vector classification as our best matcher.

Iteration 6 (Debugging)

Instead of debugging on all 6 of the classifiers, we will instead just debug on the top 2 classifiers. Below are the precision and recall of each of the classifiers, having train on training set and test on test set (training set and test set are being splitted from development set in the ratio of 50:50).

num	Classifier	Precision	Recall	F1
1.	Decision Tree	0.9166666666666666	0.9016393442622951	0.9090909090909091
2.	Logistic Regression	0.9666666666666667	0.9508196721311475	0.9586776859504132
3.	Random Forest	0.9672131147540983	0.9672131147540983	0.9672131147540983
4.	Support Vector Machine	0.9827586206896551	0.9344262295081968	0.9579831932773109
5.	Naive Bayes	0.9384615384615385	1.0	0.9682539682539683
6.	Linear Support Vector Classification	0.9838709677419355	1.0	0.991869918699187

Below are **ALL** of the false positive and false negative examples
1.

ltable: ['7633', 'Grudge Match', 'Comedy,Sport', '113', '64', '47183', 'Peter Segal', '2013']

rtable: ['8541', 'Grudge Match', 'Comedy,Drama,Sports,Fitness', '113', '64', '35912', 'Peter Segal,', '2013']

Label: 1 Predicted: 0

2.

ltable: ['13888', 'Starman', 'Romance,Sci-Fi', '115', '70', '33597', 'John Carpenter', '1984']

rtable: ['8208', 'Starman', 'Drama,Science Fiction,Fantasy,Romance', '115', '66', '24445', 'John Carpenter,', '1984']

Label: 1 Predicted: 0

3.

ltable: ['16136', 'Katy Perry: Part of Me', 'Documentary,Musical', '93', '59', '12440', 'Dan Cutforth,Jane Lipsitz', '2012']

rtable: ['11957', 'Katy Perry: Part Of Me Fan Sneaks', 'Documentary', 0, '92', '72', 'Dan Cutforth,Jane Lipsitz,', '2012']

Label: 0 Predicted: 1

Problem: Katy Perry: Part of Me and Katy Perry: Part of Me Fan Sneaks are two totally different shows. One is a show while another one is like a sneak peek. However, there seem to be no way we can let the learning algorithm to know about this.

4.

ltable: ['727', 'Underdog', 'Animation,Family,Comedy', '0', '75', '1253', '', '1964']

rtable: ['2833', 'Underdog', 'Action,Adventure,Comedy,Kids,Family', '80', '60', '122557', 'Frederik Du Chau,', '2007']

Label: 0 Predicted: 1

Problem: Underdog in ltable and Underdog in rtable are two different movies. Observe the year. The problem is the learning algorithm didn’t put enough weightage on year.

Solutions:

There's pretty much nothing we can do here anymore. Since the weight of each feature is being determined by the learning algorithm, and that the learning algorithm is a black box to us, we can't change the algorithm to place more weightage or emphasis on certain feature.

In addition to that, given two movies, we can't create any feature that can tell if a movie is a sequel or an episode to another. This is because not all movies that have sequel have the character I, II, 1, 2 and so on in its title. In addition to that, it may be the case that the sequel movies are in the same year and have the same directors and actors. Therefore, it's extremely difficult for us to create a feature that can fulfill this purpose.

Compute the accuracy of our best matcher

Now that we have the best matcher as linear support vector, let's train all 6 of our classifiers on the development set and test it on the evaluation set that we haven't use until now.

Num.	Classifier	Precision	Recall	F1
1.	Decision Tree	0.9117647058823529	1.0	0.9538461538461539
2.	Logistic Regression	1.0	1.0	1.0
3.	Random Forest	0.96875	1.0	0.9841269841269841
4.	Support Vector Machine	1.0	1.0	1.0
5.	Naive Bayes	0.96875	1.0	0.9841269841269841
6.	Linear Support Vector Classification	1.0	1.0	1.0

Final feature set

The final feature set that we are using are movie title, movie genres, movie directors, duration and year. For movie title, movie genres and movie directors, we use Jaccard measure to implement our feature, tokenized by space, comma and comma respectively. For our duration and year, we implement it by return 1 if two variables are exactly the same, and 0 if otherwise. The features that we have dropped during the iterations are rating and rating count. This is because we find that this 2 attributes do not help us in determining whether two given movies are the same movie at all. Our precision improved significantly after dropping rating and rating count.

Time taken to label the data

We extract the data that we need to label from the setC.csv using seed==10. We then assign one group member to single handedly label all of the data. The time taken to label is approximated to be 3 hours.

Time taken to find the best learning based matcher

8 hours

Why can't we reach higher precision, recall and F1

We already reached higher precision, recall and F1, having 1.0 for three of them respectively. However, we recognised that we did not address the problem of creating feature vectors that can determine if two movies are sequel and the problem of setting weightage on certain attributes over another.

The reason we can't do this is because the learning algorithm is like a black box to us. The most we can do is to introduce new learning algorithm that we think will be suitable for matching data like this, and we already did that by introducing linear support vector classification.

Please note that we have also included our Jupyter notebook describing our end to end process of doing this phase.