Project Report on OSDA

Yuan Wei

Project Based on Titanic Survivor Predictions

Overview

The data has been split into two groups:
training set (train.csv)
test set (test.csv)
The training set should be used to build your machine learning models. For the training set, we provide the outcome (also known as the "ground truth") for each passenger. Your model will be based on "features" like passengers' gender and class. You can also use feature engineering to create new features.
The test set should be used to see how well your model performs on unseen data. For the test set, we do not provide the ground truth for each passenger. It is your job to predict these outcomes. For each passenger in the test set, use the model you trained to predict whether or not they survived the sinking of the Titanic.

Variable	Definition	Key	Variable
survival	Survival	0 = No, 1 = Yes	survival
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd	pclass
sex	Sex		sex
Age	Age in years		Age
sibsp	# of siblings / spouses aboard the Titanic		sibsp
parch	# of parents / children aboard the Titanic		parch
ticket	Ticket number		ticket
fare	Passenger fare		fare
cabin	Cabin number		cabin
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton	embarked

1. Basic overview
PassengerId passenger id
Training set 891 (1- 891), test set 418 (892 - 1309)
Survived Whether rescued or not

1=yes, 2=no
Rescued: 38%
Stricken: 62% (actual % in distress: 67.5%)
Pclass Ticket class
Represents socio-economic status. 1=advanced, 2=intermediate, 3=low
1 : 2 : 3 = 0.24 : 0.21 : 0.55
Name Name
Example: Futrelle, Mrs. Jacques Heath (Lily May Peel)
Example: Heikkinen, Miss. Laina
Sex Gender
male male 577, female female 314
Male : Female = 0.65 : 0.35
Age Age (missing 20% of data)
Training set: 714/891 = 80%
Test set: 332/418 = 79%
SibSp Total number of siblings or spouses of peers
68% none, 23% have 1 ... max 8
Parch Total number of parents or children in the same row
76% none, 13% with 1, 9% with 2 ... max 6
Some children travelled only with a nanny, therefore parch=0 for them.
Ticket Ticket number (format not standardised)
Example: A/5 21171
Example: STON/O2. 3101282
Fare
The test set is missing one data
Cabin number
The training set has only 204 data, the test set has 91 data
Example: C85
Embarked Port of embarkation
C = Cherbourg 19%, Q = Queenstown 9%, S = Southampton 72%
Training set with two less data

2. Exploring the data

2.1 Basic information on features (head, info, describe)

```
# Explore the data
# View field structures, types and head examples
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId     891 non-null int64
Survived        891 non-null int64
Pclass          891 non-null int64
Name            891 non-null object
Sex             891 non-null object
```

```
Age              714 non-null float64
SibSp            891 non-null int64
Parch            891 non-null int64
Ticket           891 non-null object
Fare             891 non-null float64
Cabin            204 non-null object
Embarked         889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
PassengerId      418 non-null int64
Pclass           418 non-null int64
Name             418 non-null object
Sex              418 non-null object
Age              332 non-null float64
SibSp            418 non-null int64
Parch            418 non-null int64
Ticket           418 non-null object
Fare             417 non-null float64
Cabin            91 non-null object
Embarked         418 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

2.2 Correlation of several enumerated features with Survived (direct group aggregation for mean value)

Higher survival rates for the rich and middle classes, lower survival rates for the bottom

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

Gender and survival are strongly correlated, with female users having a significantly higher survival rate than males

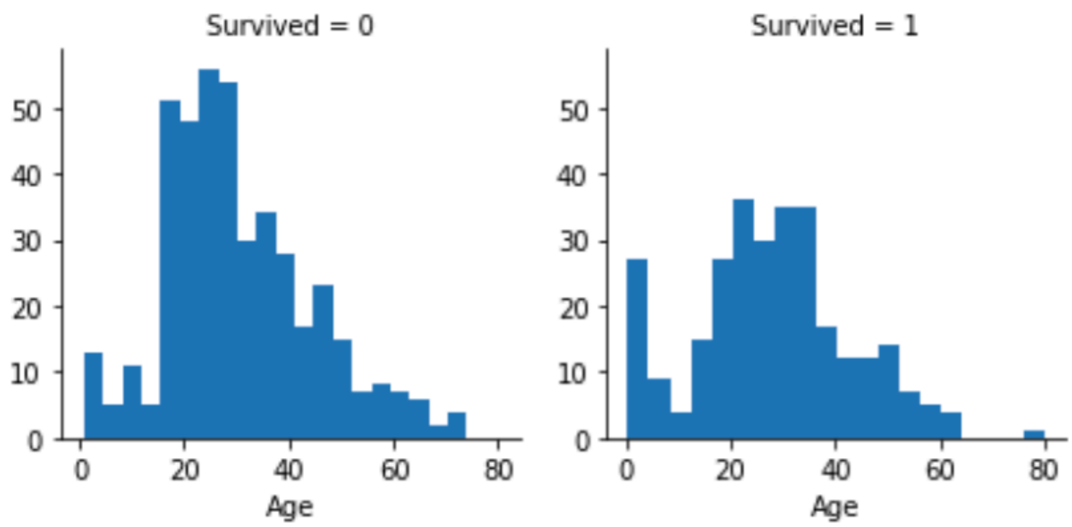
	Sex	Survived
0	female	0.742038
1	male	0.188908

The chances of surviving with 0 to 2 siblings or spouses are higher than with more

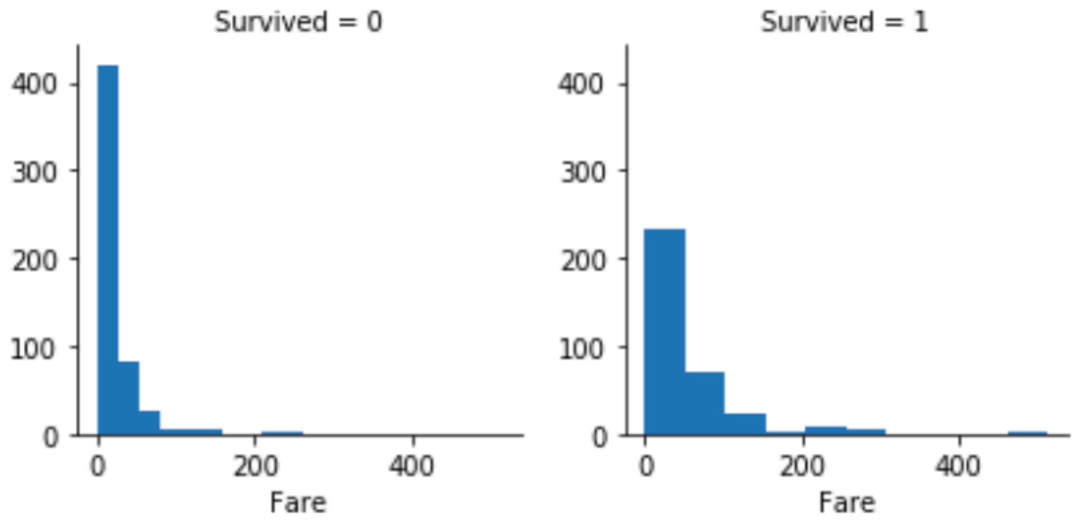
	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

2.3 Separate histograms were used to look at the distribution of survivorship and non-survivorship for characteristics with a long span such as age

Greater chance of survival for babies and young children



Low chance of survival for the cheapest ticket



3. Feature cleaning

3.1 HasCabin

#Removal of the features Ticket (no human judgement correlation) and Cabin (too little valid data)

3.2 Title

Create a designation feature based on the name, which will contain gender and class information

dataset.Name.str.extract(' ([A-Za-z]+)\.' -> Extract strings starting with space . Ending strings are extracted

Match with gender to see if each type of title belongs to male or female for subsequent categorization

Categorise titles as Mr,Miss,Mrs,Master,Rare_Male,Rare_Female (with Rare differentiated by male and female)

Summarize Survived Means by Title to see correlation

Sex	female	male
Title		
Mr	0	517
Master	0	40
Dr	1	6
Rev	0	6
Col	0	2
Major	0	2
Capt	0	1
Don	0	1
Jonkheer	0	1
Sir	0	1
Miss	182	0
Mrs	125	0
Mlle	2	0
Countess	1	0
Lady	1	0
Mme	1	0
Ms	1	0

	Title	Survived
0	Master	0.575000
1	Miss	0.704301
2	Mr	0.156673
3	Mrs	0.792000
4	Rare_Female	1.000000
5	Rare_Male	0.285714

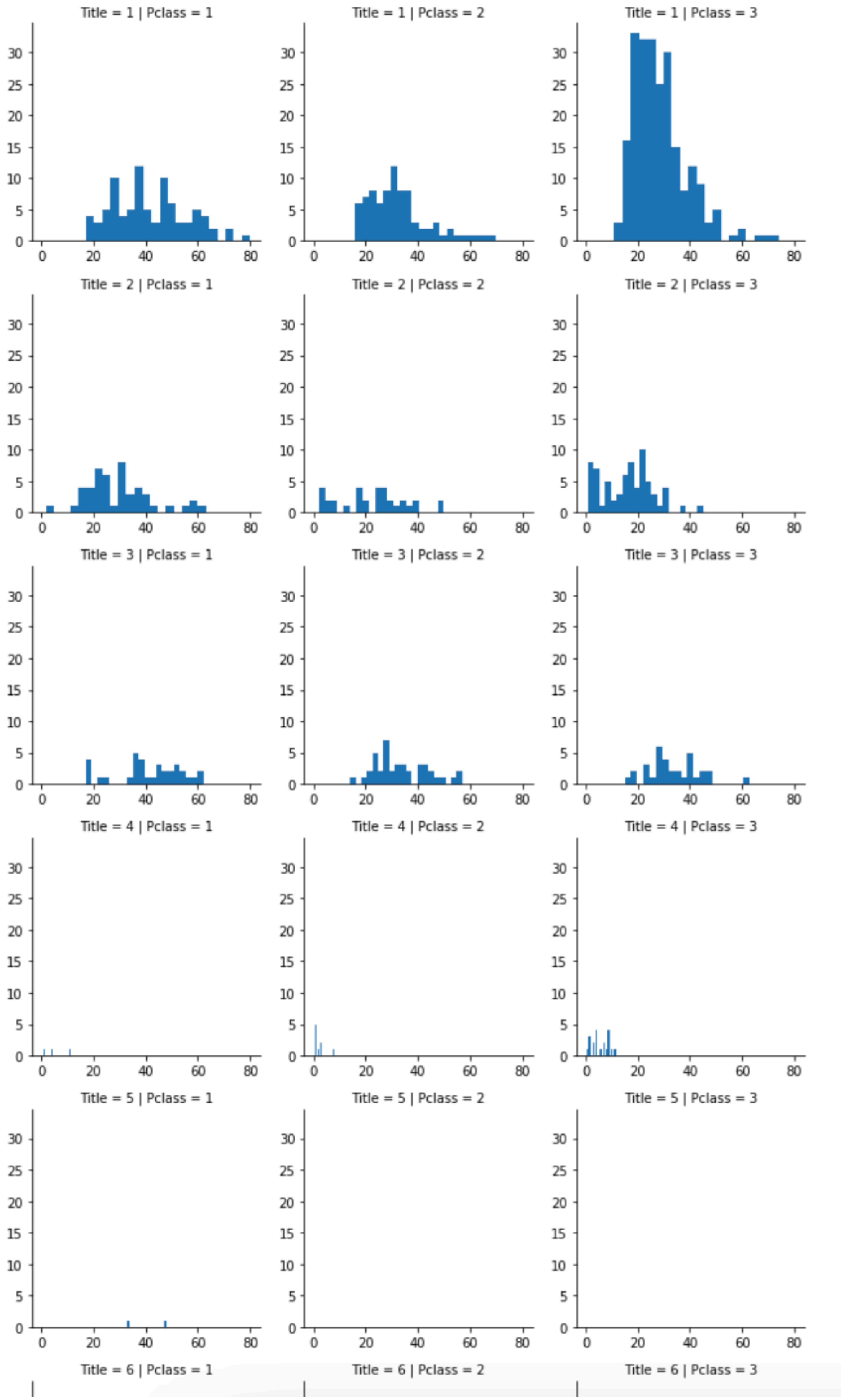
3.3 Sex

Sex features mapped to values

	Survived	Pclass	Sex
0	0	3	0
1	1	1	1
2	1	3	1
3	1	1	1
4	0	3	0

3.4 Age

Predictive supplementation for null values in the Age field
Take the median age of the same Pclass and Title for supplementation (Demo for Pclass and Sex)


Title = 1 | Pclass = 1
Title = 1 | Pclass = 2
Title = 1 | Pclass = 3
Title = 2 | Pclass = 1
Title = 2 | Pclass = 2
Title = 2 | Pclass = 3
Title = 3 | Pclass = 1
Title = 3 | Pclass = 2
Title = 3 | Pclass = 3
Title = 4 | Pclass = 1
Title = 4 | Pclass = 2
Title = 4 | Pclass = 3
Title = 5 | Pclass = 1
Title = 5 | Pclass = 2
Title = 5 | Pclass = 3
Title = 6 | Pclass = 1
Title = 6 | Pclass = 2
Title = 6 | Pclass = 3

Populate valuation with empty values for the age field
Use the median Age of the same Pclass and Title instead (for combinations where the median is empty, use the median of the Title as a whole instead)

3.4 IsChildren
Create whether **c**hild features based on age

Fare	Embarked	NameLength	HasCabin	Title	IsChildren
'.2500	S	23	0	1	0.0
'1.2833	C	51	1	3	0.0
'.9250	S	22	0	2	0.0
;3.1000	S	44	1	3	0.0
;.0500	S	24	0	1	0.0

Create age interval features
pd.cut is a uniform cut by the size of the value, each set of value intervals is the same size, but the number of samples may not be the same
pd.qcut is cut by frequency of distribution of samples over values, same number of samples per group

	Survived	Pclass	Sex	Age	SibSp	Parch
0	0	3	0	2	1	0
1	1	1	1	6	1	0
2	1	3	1	3	0	0
3	1	1	1	5	1	0
4	0	3	0	5	0	0

3.5 Family Size
Create family size FamilySize portfolio features

	FamilySize	Survived
0	1	0.303538
1	2	0.552795
2	3	0.578431
3	4	0.724138
4	5	0.200000
5	6	0.136364
6	7	0.333333
7	8	0.000000
8	11	0.000000

3.6 IsAlone

Create whether the IsAlone feature is alone

	IsAlone	Survived
0	0	0.505650
1	1	0.303538

3.7 Embarked

Get the most ports of embarkation

	Embarked	Survived
0	C	0.553571
1	Q	0.389610
2	S	0.339009

3.8 Fare

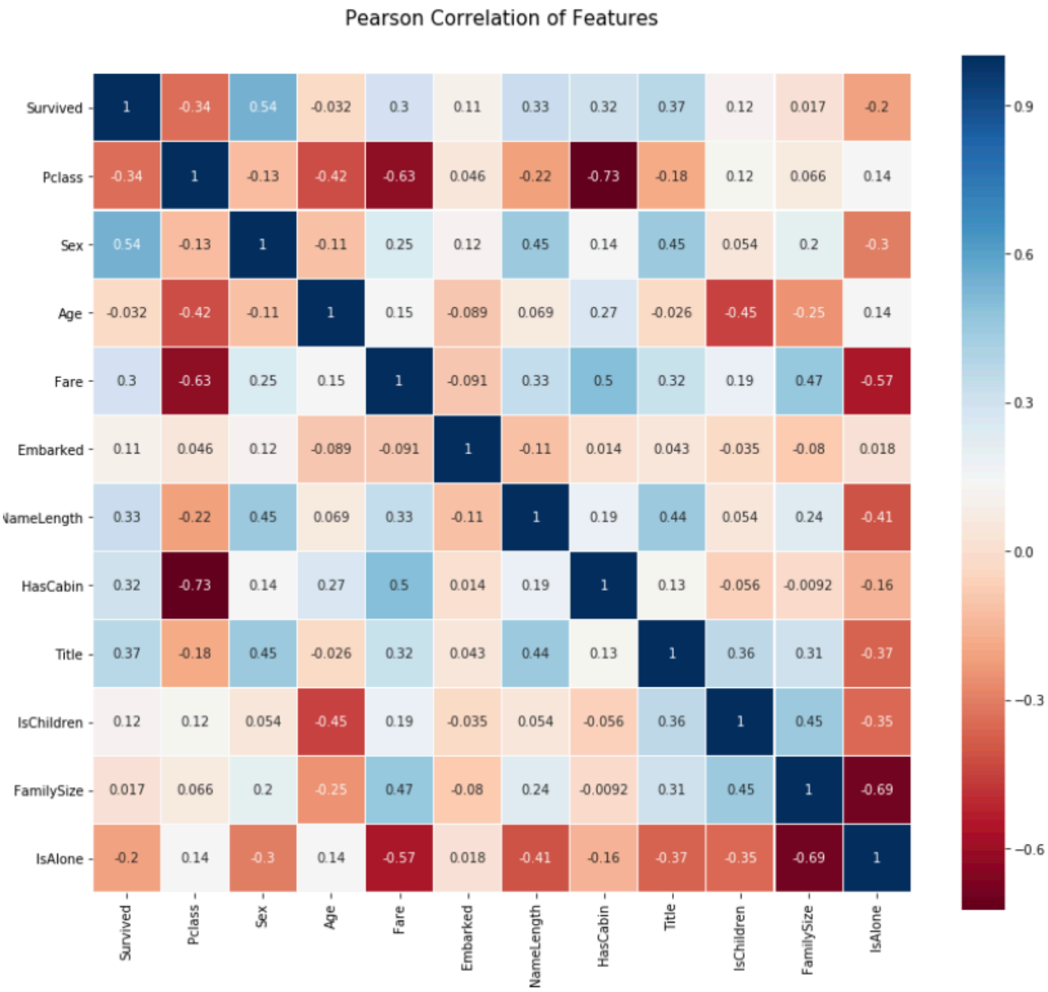
Fill the test set with null values for Fare, using the median

Convert Fare features to ordinal values based on FareBand

	FareBand	Survived
0	(-0.001, 7.91]	0.197309
1	(7.91, 14.454]	0.303571
2	(14.454, 31.0]	0.454955
3	(31.0, 512.329]	0.581081

3.9 Feature correlation visualization¶

Visualization of correlation between features using seaborn's heatmap



3.10 Processed Dataset

3.10.1 New Features:

```
 #    Column        Non-Null Count  Dtype
---   ------        --------------  -----
 0    PassengerId   418 non-null    int64
 1    Pclass        418 non-null    int64
 2    Sex           418 non-null    int64
 3    Age           418 non-null    int64
 4    Fare          418 non-null    float64
 5    Embarked      418 non-null    int64
 6    HasCabin      418 non-null    int64
 7    Title         418 non-null    int64
 8    IsChildren    418 non-null    float64
 9    FamilySize    418 non-null    int64
 10   IsAlone       418 non-null    int64
```

3.10.2 New look

	Survived	Pclass	Sex	Age	Fare	Embarked	HasCabin	Title	IsChildren	FamilySize	IsAlone
0	0	3	0	1	0	1	0	1	0.0	3	0
1	1	1	1	1	3	2	1	3	0.0	3	0
2	1	3	1	0	1	1	0	2	0.0	3	1
3	1	1	1	2	3	1	1	3	0.0	3	0
4	0	3	0	2	1	1	0	1	0.0	3	1
...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	0	1	1	1	0	6	0.0	3	1
887	1	1	1	1	2	1	1	2	0.0	3	1
888	0	3	1	1	2	1	0	2	0.0	4	0
889	1	1	0	0	2	2	1	1	0.0	3	1
890	0	3	0	2	0	0	0	1	0.0	3	1

3.10.3 Binarization

Using proposed algorithm:

```
def binarize_X(X: pd.DataFrame) -> 'pd.DataFrame[bool]':
    """Scale values from X into pandas.DataFrame of binary values"""
    dummies = [pd.get_dummies(X[f], prefix=f, prefix_sep=': ') for f in X.columns]
    X_bin = pd.concat(dummies, axis=1).astype(bool)
    return X_bin
```

	Pclass: 1	Pclass: 2	Pclass: 3	Sex: 0	Sex: 1	Age: 0	Age: 1	Age: 2	Fare: 0	Fare: 1	...	Title: 5	Title: 6	IsChildren: 0.0	IsChildren: 1.0
0	False	False	True	True	False	False	True	False	True	False	...	False	False	True	False
1	True	False	False	False	True	False	True	False	False	False	...	False	False	True	False
2	False	False	True	False	True	True	False	False	False	True	...	False	False	True	False
3	True	False	False	False	True	False	False	True	False	False	...	False	False	True	False
4	False	False	True	True	False	False	False	True	False	True	...	False	False	True	False

4. Modelling and optimisation

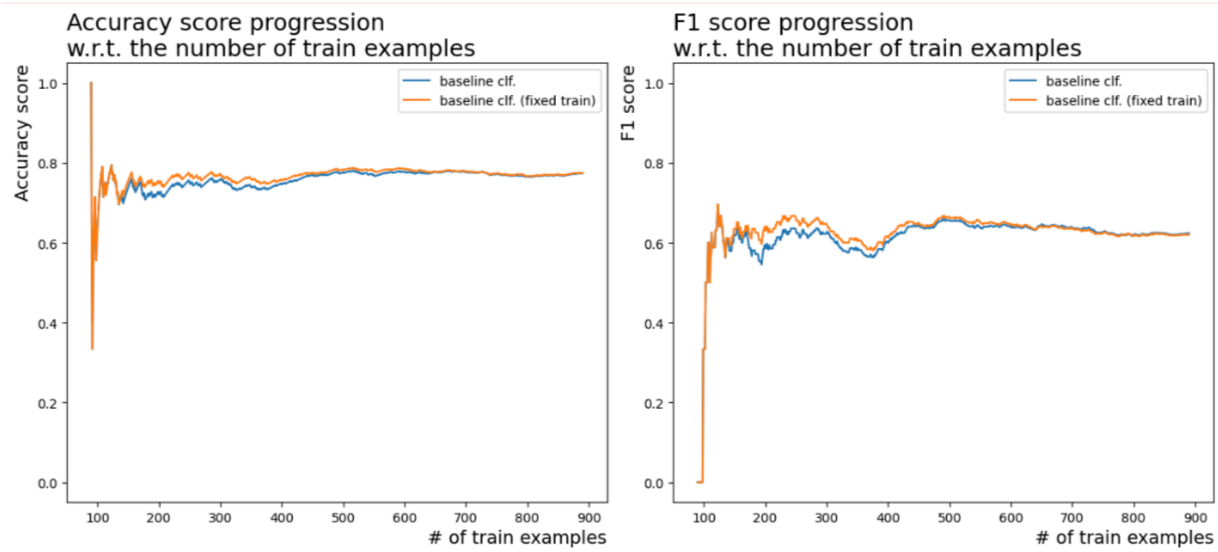
4.1 Model comparison

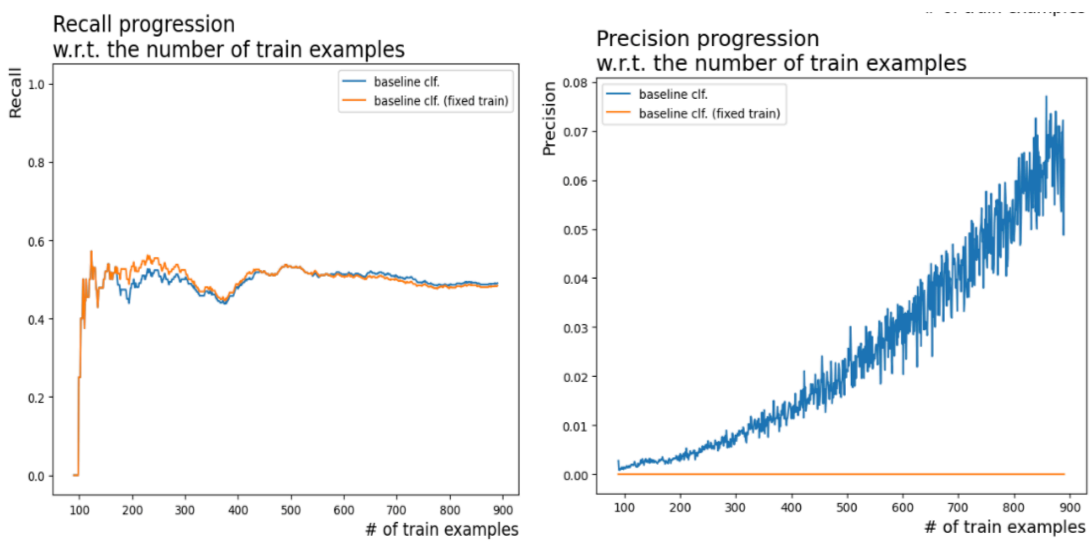
4.1.1 LazyFCA

Acc = 79.05

Precision = 0.82

Recall = 0.49





Here we can see that there is a wide gap between the accuracy and recall of LazyFCA, and the resulting f1 scores we obtained are not informative. Secondly there is a degree of imbalance in the data set and the accuracy rate is affected to some extent. And based on the data obtained, LazyFCA model is still relatively conservative and will still fall short of the true predictions. So here we can clearly see the lack of coverage of the real samples by looking at the recall rate.

4.1.2 Logistic Regression

Acc = 82.15

Precision = 0.76
Recall = 0.70

4.1.3 Support Vector Machines
Acc = 83.28
Precision = 0.81
Recall = 0.72

4.1.4 KNN
Acc = 85.3
Precision = 0.81
Recall = 0.67

4.1.5 Decision Tree
Acc = 88.78
Precision = 0.79
Recall = 0.69

4.1.6 Random Forest
Acc = 83.15
Precision = 0.80
Recall = 0.71

Model	Accuracy	Precision	Recall
LazyFCA	79.05	0.82	0.49
Logistic Regression	82.15	0.76	0.70
Support Vector Machines	83.28	0.81	0.72
KNN	85.30	0.81	0.67
Decision Tree	88.78	0.79	0.69
Random Forest	83.15	0.80	0.71

4.2.1 Optimisation

Baseline Algorithm

```python
X_pos = [x_train for x_train, y in zip(X_train, Y_train) if y]
X_neg = [x_train for x_train, y in zip(X_train, Y_train) if not y]

n_counters_pos = 0  # number of counter examples for positive intersections
for x_pos in X_pos:
    intersection_pos = x & x_pos
    if len(intersection_pos) < min_cardinality:  # the intersection is too small
        continue

    for x_neg in X_neg:  # count all negative examples that contain intersection_pos
        if (intersection_pos & x_neg) == intersection_pos:
            n_counters_pos += 1

n_counters_neg = 0  # number of counter examples for negative intersections
for x_neg in X_neg:
    intersection_neg = x & x_neg
    if len(intersection_neg) < min_cardinality:
        continue

    for x_pos in X_pos:  # count all positive examples that contain intersection_neg
        if (intersection_neg & x_pos) == intersection_neg:
            n_counters_neg += 1

perc_counters_pos = n_counters_pos / len(X_pos)
perc_counters_neg = n_counters_neg / len(X_neg)

prediction = perc_counters_pos < perc_counters_neg
return prediction
```

Rewrite Baseline Algorithm with Numpy

```python
X_pos = np.array([x_train for x_train, y in zip(np.array(X_train), Y_train) if y])
X_neg = np.array([x_train for x_train, y in zip(np.array(X_train), Y_train) if not y])
n_counters_pos = 0  # number of counter examples for positive intersections
for x_pos in X_pos:
    intersection_pos = x & x_pos
    if intersection_pos.sum() < min_cardinality:  # the intersection is too small
        continue
    n_counters_pos += ((intersection_pos & X_neg) == intersection_pos).all(axis=1).sum()

n_counters_neg = 0  # number of counter examples for negative intersections
for x_neg in X_neg:
    intersection_neg = x & x_neg
    if len(intersection_neg) < min_cardinality:
        continue
    n_counters_neg += ((intersection_neg & X_pos) == intersection_neg).all(axis=1).sum()

perc_counters_pos = n_counters_pos / len(X_pos)
perc_counters_neg = n_counters_neg / len(X_neg)

prediction = perc_counters_pos < perc_counters_neg
return prediction
```

Model	Running Time	Accuracy	Precision	Recall
LazyFCA	20s	75.71	0.81	0.49
LazyFCA with Numpy	11.3s	75.64	0.81	0.49

Conclusion

It is easy to see from the above that the rewritten code clearly outperforms the original algorithm in terms of running speed. However, since the core logic has not changed,

there is hardly any difference in prediction accuracy.
In comparison with the existing mainstream algorithms, lazyfca does not have a clear advantage and is far below the mainstream algorithms in terms of recall.
Decision trees clearly outperform other algorithms in predicting the dataset used in this task, while the simplest KNN also performs similarly to decision trees
If we take into account that this dataset is inherently unbalanced and we do not consider accuracy, then the support vector machine and random forest perform best.