Dell Case Study - Model Creation

August 21, 2025

1 Model Creation

1.1 Data Loading and Cleaning

[1]:
```
# Loading necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import (precision_score, recall_score, f1_score,␣
 ↪roc_auc_score,classification_report, confusion_matrix)
import matplotlib.pyplot as plt

# Loading data
df = pd.read_csv("Downloads/Learnova_Leads (1).csv")

# Data overview and checking for data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4612 entries, 0 to 4611
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   user_id                4612 non-null   object
 1   user_age               4612 non-null   int64
 2   occupation_status      4612 non-null   object
 3   initial_contact        4612 non-null   object
 4   profile_status         4612 non-null   object
 5   site_visits            4612 non-null   int64
 6   engagement_time        4612 non-null   int64
 7   avg_pages_per_session  4612 non-null   float64
 8   recent_engagement      4612 non-null   object
 9   newspaper_ad           4612 non-null   object
 10  magazine_ad            4612 non-null   object
 11  online_ad              4612 non-null   object
 12  edu_forums             4612 non-null   object
 13  word_of_mouth          4612 non-null   object
 14  enrollment_status      4612 non-null   int64
dtypes: float64(1), int64(4), object(10)
memory usage: 540.6+ KB
```

[2]:
```
# Data Cleaning - Converting "Yes/No" strings to numerical values (0/1)
for c in␣
 ↪["newspaper_ad","magazine_ad","online_ad","edu_forums","word_of_mouth"]:
    if c in df.columns and df[c].dtype == object:
        df[c] = df[c].str.strip().str.lower().map({"yes":1, "no":0})

# Re-check for data type
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4612 entries, 0 to 4611
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   user_id                4612 non-null   object
 1   user_age               4612 non-null   int64
 2   occupation_status      4612 non-null   object
 3   initial_contact        4612 non-null   object
 4   profile_status         4612 non-null   object
 5   site_visits            4612 non-null   int64
 6   engagement_time        4612 non-null   int64
 7   avg_pages_per_session  4612 non-null   float64
 8   recent_engagement      4612 non-null   object
 9   newspaper_ad           4612 non-null   int64
 10  magazine_ad            4612 non-null   int64
 11  online_ad              4612 non-null   int64
 12  edu_forums             4612 non-null   int64
 13  word_of_mouth          4612 non-null   int64
 14  enrollment_status      4612 non-null   int64
dtypes: float64(1), int64(9), object(5)
memory usage: 540.6+ KB
```

[2]:
```
  user_id  user_age occupation_status initial_contact profile_status  \
0  EXT001        57        Unemployed         Website           High
1  EXT002        56      Professional      Mobile App         Medium
2  EXT003        52      Professional         Website         Medium
3  EXT004        53        Unemployed         Website           High
4  EXT005        23           Student         Website           High
```

```
   site_visits  engagement_time  avg_pages_per_session recent_engagement  \
0            7             1639                  1.861  Website Activity
1            2               83                  0.320  Website Activity
2            3              330                  0.074  Website Activity
3            4              464                  2.057  Website Activity
4            4              600                 16.914    Email Activity

   newspaper_ad  magazine_ad  online_ad  edu_forums  word_of_mouth  \
0             1            0          1           0              0
1             0            0          0           1              0
2             0            0          1           0              0
3             0            0          0           0              0
4             0            0          0           0              0

   enrollment_status
0                  1
1                  0
2                  0
3                  1
4                  0
```

Before beginning to create any models, we have to load the data and clean it accordingly, if needed. In order for us to utilize our model for all the variables, we must convert categorical variables into numerical ones. As an initial conversion, I mapped all "Yes/No" string (object) values to 0, representing "No", and 1, representing "Yes". Here, I have double-checked that we have correctly converted the data types.

However, we are not done yet. We still have to identify our target variables by converting the enrollment status column to integers, create the feature set (removing the target variable and user ID), and preprocess any remaining categorical columns into numeric ones. We will achieve this through one-hot encoding (OHE), which essentially converts categorical variables to binary columns, using 1 as an initial baseline. Finally, we will split our dataset into training and testing sets so we can analyze our model correctly.

[3]:
```
# Setting target variable and creating feature set
y = df["enrollment_status"].astype(int)
X = df.drop(columns=["enrollment_status", "user_id"], errors="ignore")

# Identify numeric vs categorical columns
num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = X.select_dtypes(include=["object","category"]).columns.tolist()

# One-Hot Encoding
ohe = ColumnTransformer(transformers=[("num", StandardScaler(), num_cols),
        ("cat", OneHotEncoder(handle_unknown="ignore", sparse_output=False),␣
 ↪cat_cols),],remainder="drop",)

# Train/Test split sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42, stratify=y)
    # 80% training, 20% test; stratify maintains the same class distribution␣
↪for both sets
```

Now that we have all our necessary data preprocessed, and our training and test datasets split, we can start creating our models.

1.2 Defining Models

To start defining our models for use, we will create a dictionary called "models" containing 3 different model pipelines – "logistic" for Logistic Regression Model, "random_forest" for Random Forest Model, and "boosted_tree" for our Boosted Tree Model. Each has 2 steps, a "prep" step where we preprocess using the OHE, and a "clf" step, or classifying step using the appropriate model classifier algorithms.

[4]:
```
models = {
    "logistic": Pipeline(steps=[("prep", ohe),
                                ("clf", LogisticRegression(max_iter=2000,␣
↪class_weight="balanced", solver="liblinear"))
    ]),
    "random_forest": Pipeline(steps=[("prep", ohe), ("clf",␣
↪RandomForestClassifier(
                                n_estimators=400, min_samples_split=4,␣
↪min_samples_leaf=2,
                                class_weight="balanced_subsample",␣
↪random_state=42, n_jobs=-1))
    ]),
    "boosted_tree": Pipeline(steps=[("prep", ohe), ("clf",␣
↪GradientBoostingClassifier(
                                n_estimators=400, learning_rate=0.05,␣
↪max_depth=3, random_state=42))
    ]),
}
```

This structure allows for easy comparison between different classification algortihms while using the same preprocessing step.

1.3 Performance Metrics

[5]:
```
# Evaluate performance metrics
def eval_probs(y_true, p, thr=0.5):
    y_pred = (p >= thr).astype(int)
    return {
        "precision": precision_score(y_true, y_pred, zero_division=0),
        "recall":    recall_score(y_true, y_pred, zero_division=0),
        "f1":        f1_score(y_true, y_pred, zero_division=0),
        "roc_auc":   roc_auc_score(y_true, p),
    }
```

Here, we have defined a function that evaluates the classification performance metrics based on predicted probabilities. The function takes in true labels, predicted probabilities, and a threshold (which we have defaulted to 0.5). The "y_pred" part of the function converts probability predictions to binary class predictions (0 or 1), comparing each probability to the threshold and converting the binary result to integers.

Most importantly, the function returns a dictionary with evaluation/performance metrics:

- **Precision:** Measures the ratio of *true* positives to all *predicted* positives. Of the people the model said would enroll, how many actually did?
- **Recall:** Measures the ratio of *true* positives to all *actual* positives. Of the people who did enroll, how many did the model correctly identify?
- **F1-Score:** Mean or combination of precision and recall. A balance of the two.
- **ROC-AUC Score:** The area under the ROC curve, measuring distinguishing ability. How well does the model rank positive examples higher than negative ones?

1.4 Model Implementation and Threshold Tuning

[6]:
```python
# Fitting, predicting, and scoring
results = {}
for name, pipe in models.items():
    pipe.fit(X_train, y_train)
    proba = pipe.predict_proba(X_test)[:, 1]  # "propensity to enroll" score
    metrics_50 = eval_probs(y_test, proba, thr=0.50)  # default threshold 0.50

    # Tuning thresholds to maximize F1
    thresholds = np.linspace(0.1, 0.9, 81)
    f1s = []
    for t in thresholds:
        f1s.append((t, f1_score(y_test, (proba >= t).astype(int),␣
 ↪zero_division=0)))
    best_thr, best_f1 = max(f1s, key=lambda x: x[1])
    metrics_best = eval_probs(y_test, proba, thr=best_thr)

    results[name] = {
        "default(0.50)": metrics_50,
        "best_f1": {"threshold": float(best_thr), **metrics_best},
    }
```

Here, I have now implemented the model and done some threshold optimization for our models. The code iterates through each model in the "models" dictionary we created earlier, using a for loop. For each model, it fits the model to the training data, generates probability predictions on the test data (extracting positive class probabilities), and evaluating the model using the default threshold (0.50).

Then, we are able to tune the thresholds to maximize the F1-score. For each threshold created, it calculates the F1 score by converting the probabilities to binary predictions. This is all done to find the threshold that produces the highest F1 score, so the one that *balances the precision and recall metrics the best.*

2 Model Results

2.1 Performance Metrics

[7]:
```
# Organizing performance metric results into a neat table for better␣
↪readability. (ChatGPT helped a lot)
def results_to_tables(results, round_to=3):
    rows = []
    for model, settings in results.items():
        for setting_name, metrics in settings.items():
            thr = metrics.get("threshold", np.nan)
            rows.append({
                "Model": model,
                "Setting": setting_name,
                "Threshold": thr,
                "Precision": metrics["precision"],
                "Recall": metrics["recall"],
                "F1": metrics["f1"],
                "ROC_AUC": metrics["roc_auc"]
            })

    df_long = pd.DataFrame(rows)
    # Rounding numeric columns
    num_cols = ["Threshold","Precision","Recall","F1","ROC_AUC"]
    df_long[num_cols] = df_long[num_cols].apply(pd.to_numeric, errors="coerce")
    df_long[num_cols] = df_long[num_cols].round(round_to)

    # Make a compact table: one row per model, default vs best_f1
    df_wide = (
        df_long
        .pivot_table(index="Model", columns="Setting",
                     values=["Threshold","Precision","Recall","F1","ROC_AUC"])
        .loc[:, ["Threshold","Precision","Recall","F1","ROC_AUC"]]
        .sort_values(("F1","best_f1"), ascending=False)    # sort by best_f1 F1
    )

    # Neat column names
    df_wide.columns = [f"{metric} ({setting})" for metric, setting in df_wide.
↪columns]
    df_wide = df_wide.reset_index()

    return df_long, df_wide
```

```
df_long, df_wide = results_to_tables(results)

# Displaying
(
    df_wide.style
    .format(precision=3)
    .hide(axis="index")
    .set_caption("Model Performance Metrics: Default vs Best F1 threshold")
)
```

```
[7]: <pandas.io.formats.style.Styler at 0x24f594ca030>
```

Here, we have a table containing the results for our performance metrics. This gives us very intersting insights. The columns displaying "best_f1" means that it is using the Best F1 threshold to generate its result (seen in the Threshold column), and I have ranked all our values by F1-Score (best f1 threshold) so that we have a clearer idea of which model is performing the best. I will explain this in detail by looking at the Random Forest model's result (best performance results), with a business context:

Threshold

First, we will discuss our "best" threshold, which is at 0.44, a decrease from the default of 0.50. By comparing the Precision and Recall values for the "best_f1" vs. the "default", we can see that we have decreased Precision and increased Recall, essentially meaning that we view missing a potential enrollee as costly.

F1-Score

Our F1-Score, the balance of both Precision and Recall, means we can best find enrollees without wasting resources. So, we can see that our standard approach (0.5 threshold) results in an F1=0.769, while the optimized approach (0.44 threshold) results in an F1=0.780. This means:

1. The marketing team can more efficiently allocate resources.
2. We can identify slightly more potential enrollees while reducing wasted outreach.
3. Out of thousands of potential enrollees and leads, even this small improvement could mean dozens more enrollments or significant cost savings.

ROC-AUC

We've already discussed what this means from an analytical standpoint. In a business context, the ROC-AUC score is extremely important, and an exceptional indicator for prediction performance and accuracy. A score of 0.5 means the model is no better than random guessing who will enroll, while a score of 1.0 essentially means a perfect prediction. Ours has an ROC-AUC score of 0.924. This tells us that there's a 92.4% chance that the model will rank a randomly chosen *actual* enrollee higher than a randomly chosen *non-enrollee*. So, a higher ROC-AUC allows the marketing team to be more confident in the model's ability to prioritize leads.

2.2 Feature Importance

```
[8]: # Logistic coefficients (top positive/negative)
if "logistic" in models:
    log_model = models["logistic"]
    # Refit on all training to extract names
    log_model.fit(X_train, y_train)
    # Recover feature names after OHE
    ohe = log_model.named_steps["prep"].named_transformers_["cat"]
    cat_names = ohe.get_feature_names_out(cat_cols) if len(cat_cols) else np.
 ↪array([])
    feat_names = list(num_cols) + list(cat_names)
    coefs = log_model.named_steps["clf"].coef_.ravel()
    coef_df = pd.DataFrame({"feature": feat_names, "coef": coefs}).
 ↪sort_values("coef", ascending=False)
    print("\nTop Positive Factors for Enrollment:\n", coef_df.head(10))
    print("\nTop Negative Factors for Enrollment:\n", coef_df.tail(10))
```

```
Top Positive Factors for Enrollment:
                                 feature      coef
14                   profile_status_High  0.974011
13               initial_contact_Website  0.961373
2                        engagement_time  0.939753
9         occupation_status_Professional  0.533432
19  recent_engagement_Website Activity  0.458053
8                           word_of_mouth  0.258032
0                                user_age  0.090403
5                             magazine_ad  0.037023
6                               online_ad  0.026679
1                             site_visits  0.016431

Top Negative Factors for Enrollment:
                                 feature      coef
7                              edu_forums  0.016188
11        occupation_status_Unemployed  0.015365
4                            newspaper_ad  0.005546
3                  avg_pages_per_session -0.013491
17  recent_engagement_Email Activity -0.165510
16                 profile_status_Medium -0.585466
18  recent_engagement_Phone Activity -0.842769
15                    profile_status_Low -0.938772
10           occupation_status_Student -1.099024
12            initial_contact_Mobile App -1.511600
```

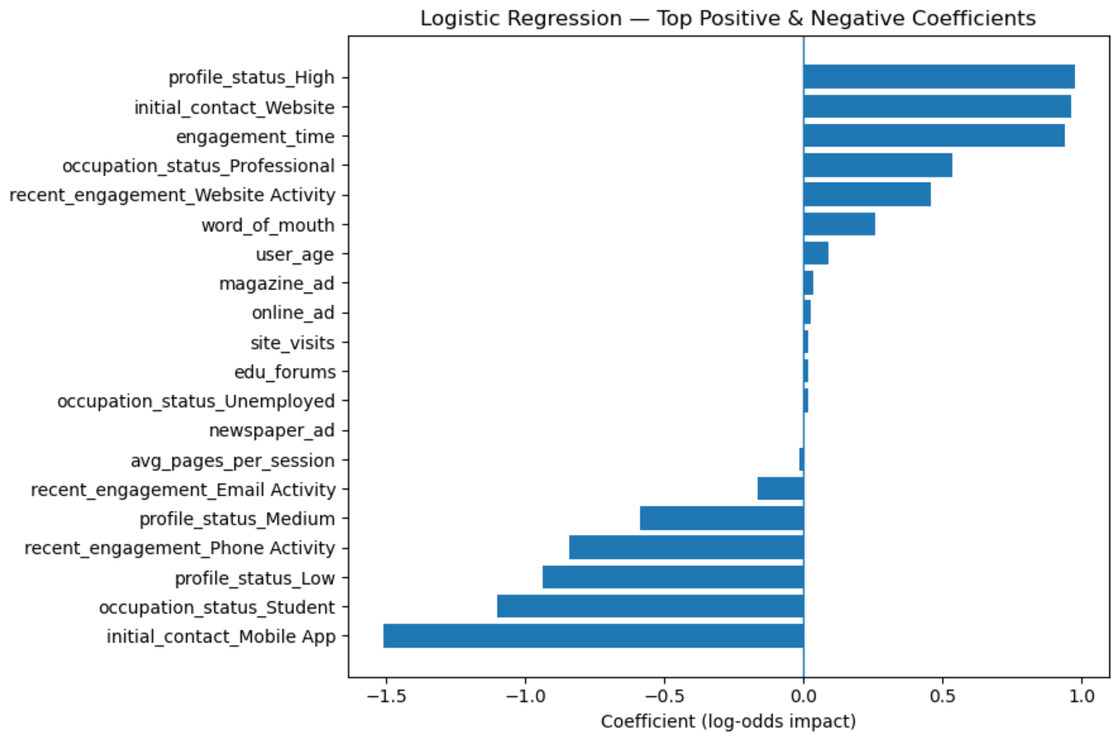
Looking at these two lists, the Logistics Regression model gives us very valuable and interpretable insights, displaying the most influential features – interpretability being exactly what this model is best at. By extracting the feature names from the OHE (for categorical variables) and numerical variables, it produces coefficients, essentially representing the change in the log odds of the target variable (Enrollment Status) for a one-unit increase. In other words, *it shows which features are strongest predictors for whether someone enrolls or not.*

For the Top Positive Factors, we can see that our initial and refined insights are backed by our model: Profile Status (High), Initial Contact (through Website), and Engagement Time are the most significant factors that drive enrollment – by quite a large margin. We can see that other features, such as Website Activity, Occupation Status (Professional), and Word of Mouth are also great factors for enrollment, factors that I've previously highlighted and explained in my case study presentation.

Again, for the Top Negative Factors, our insights are backed by the model, as well as other interesting insights that we can analyze. According to the Logistics Regression model, features such as Initial Contact (through Mobile App), Occupation Status (Student), Profile Status (Low), and Recent Engagement (Phone) are the worst factors for enrollment, meaning the predicted probability that someone who has these factors enrolls, is low. These are all factors I've discussed previously in my presentation. App downloads can skew data (download now but decide later, or never), with more friction to complete enrollment/payment. Students face doubt about enrolling due to cost decisions and sensitive schedules. People who have a low profile status simply aren't interested. And engaging over the phone comes in colder segments later down in the outreach sequence that I have highlighted in my previous findings, so it's not a priority.

[9]:
```
# Isolating top-10 positive and negatives
top_neg = coef_df.head(10)
top_pos = coef_df.tail(10)
top_coefs = pd.concat([top_neg, top_pos]).sort_values("coef")

# Plotting
fig, ax = plt.subplots(figsize=(9, 6))
ax.barh(top_coefs["feature"], top_coefs["coef"])
ax.axvline(0, linewidth=1)
ax.set_title("Logistic Regression - Top Positive & Negative Coefficients")
ax.set_xlabel("Coefficient (log-odds impact)")
plt.tight_layout()
plt.show()
```



This graph gives us a nice visual idea of how the different features stack up to each other.

```
[10]: # RF feature importances (after preprocessing)
if "random_forest" in models:
    rf_model = models["random_forest"]
    # Already fitted; recover names
    ohe = rf_model.named_steps["prep"].named_transformers_["cat"]
    cat_names = ohe.get_feature_names_out(cat_cols) if len(cat_cols) else np.
 ↪array([])
    feat_names = list(num_cols) + list(cat_names)
    importances = rf_model.named_steps["clf"].feature_importances_
    imp_df = pd.DataFrame({"feature": feat_names, "importance": importances}).
 ↪sort_values("importance", ascending=False)
    print("\nRandom Forest Top Factors of Enrollment Status:\n", imp_df.
 ↪head(20))
```

```
Random Forest Top Factors of Enrollment Status:
                                feature  importance
2                      engagement_time    0.274983
13             initial_contact_Website    0.119054
12          initial_contact_Mobile App    0.116373
0                             user_age    0.079260
3                 avg_pages_per_session    0.077584
```

```
14                 profile_status_High    0.066693
16               profile_status_Medium    0.063734
1                          site_visits    0.046182
18    recent_engagement_Phone Activity    0.029504
9       occupation_status_Professional    0.022441
19  recent_engagement_Website Activity    0.021826
10           occupation_status_Student    0.017279
17    recent_engagement_Email Activity    0.015615
11        occupation_status_Unemployed    0.013542
7                           edu_forums    0.006987
8                         word_of_mouth    0.006676
6                            online_ad    0.006449
4                         newspaper_ad    0.006154
15                  profile_status_Low    0.005733
5                          magazine_ad    0.003932
```

Similarly, we have a list of Top Factors for Enrollment based on our Random Forest Model. This essentially shows us the same information and insights as the Logistic Regression model – the higher the importance, the better the feature is at predicting enrollment. Much like the Logistic Regression's key features, the top features here are Engagement Time, Initial Contact, and Profile Status. However, a key difference here is that this shows importance for enrollment in general, both positive and negative. So, the Initial Contact for Mobile App users has a high importance because it has a very high negative impact on predicting enrollment (predicts that one will NOT enroll).

Other reasons why these features can differ between Logistic and Random Forest include that Random Forests can capture complex non-linear patterns that Logistic Regression can't, Logistic Regression shows both direction and magnitude while Random Forest shows only magnitude, and Random Forests automatically consider interactions between different features.

Additionally, it is important to reiterate that these values show correlation, not causation. A high importance value means the feature is useful for prediction, not necessarily that it causes the outcome.

3 Conclusion

After creating our model and implementing it, we can determine the best model to use based on the performance metrics. I have chosen the **Random Forest Model** as the best model choice, as it has the highest F1-Score and ROC-AUC. However, the Logistics Regression Model is still relevant, as we can use it for its easy interpretability (analyzing the top positive and negative factors).

3.1 Actionable Recommendations

Our recommendations stand firm:

1. **Prioritize Hot Leads**
2. **Deploy Predictive Scoring**
3. **Outreach Sequencing**
4. **Referral Programs**
5. **Pilot Test**

Maybe the most imporant recommendation is deploying the predictive scoring for our leads, ranking them by predicted enrollment probability. We have powered this using our Random Forest Model, which we have established as our best model. Below, we have our Top 25 leads that we should prioritize and reach out to first. This will allow us to jumpstart our outreach sequencing and prioritization of leads, all part of the 2-week plan I have proposed.

[11]:
```
# Probability of Enrollment Scores: ALL leads in the dataset, ONLY those who␣
↪HAVEN'T ENROLLED yet
rf = models["random_forest"]
rf.fit(X_train, y_train)

# Score ONLY non-enrolled leads (across entire dataset, not just test set)
mask_all_not_enrolled = (y == 0)
X_to_score = X.loc[mask_all_not_enrolled].copy()

# Using user-ID column
id_col = "user_id" if "user_id" in df.columns else None
lead_id = (
    df.loc[X_to_score.index, id_col]
    if id_col else pd.Series(X_to_score.index, index=X_to_score.index,␣
↪name="row_index")
)

# Building scores table (Random Forest only)
scores_rf = pd.DataFrame({
    "lead_id": lead_id.values,
    "propensity_random_forest": rf.predict_proba(X_to_score)[:, 1]
}).sort_values("propensity_random_forest", ascending=False).
↪reset_index(drop=True)

# Displaying table - top 25
display(
    scores_rf.head(25)
            .style.format({"propensity_random_forest": "{:.3f}"})
            .hide(axis="index")
            .set_caption("Random Forest: Probability to Enroll (Non-Enrolled␣
↪Leads, Top 25)")
)
```

```
<pandas.io.formats.style.Styler at 0x24f5c6362d0>
```

[13]:
```
pd.options.display.float_format = "{:.3f}".format
display(scores_rf.head(25))
```

```
    lead_id  propensity_random_forest
0    EXT482                     0.941
1   EXT3288                     0.929
2   EXT3112                     0.919
```

3	EXT1759	0.918
4	EXT718	0.917
5	EXT1385	0.903
6	EXT1830	0.894
7	EXT2402	0.893
8	EXT2901	0.883
9	EXT1841	0.876
10	EXT3981	0.874
11	EXT4029	0.868
12	EXT2234	0.867
13	EXT4199	0.865
14	EXT4533	0.864
15	EXT710	0.864
16	EXT444	0.863
17	EXT1408	0.845
18	EXT2320	0.834
19	EXT1253	0.830
20	EXT4009	0.826
21	EXT1404	0.814
22	EXT3067	0.810
23	EXT2045	0.797
24	EXT969	0.794