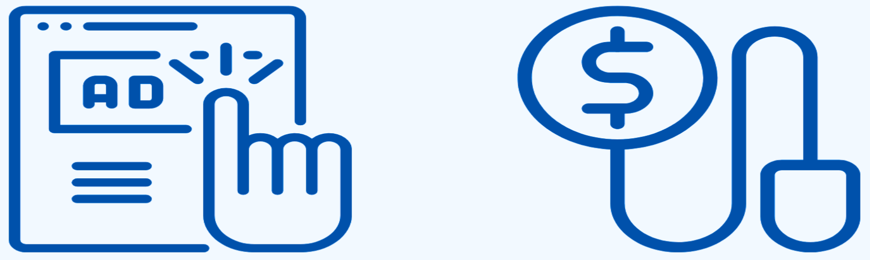
AdTracking Fraud Detection

Team Members
Janit Bidhan
Sreenivasa Rayaprolu

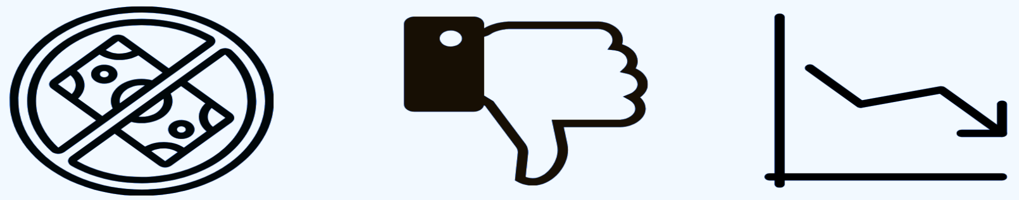
CLICK FRAUD!

Click fraud is a form of marketing fraud that occurs when pay-per-click (PPC) online ads are illegally clicked to increase site revenue or exhaust a company's budget. It is often intentional, malicious, and has no potential for clicks to result in a sale.

CLICK FRAUD CAN OCCUR BETWEEN ADVERTISERS, BETWEEN PUBLISHERS, AS VANDALISM, AS SEARCH RESULT MANIPULATION, AND MORE.



Click Fraud Can Cost Your Business



EXHAUSTS YOUR BUDGET WITH LITTLE TO NO RESULTS

LOSING TRUST AND RELATIONSHIPS WITH ADVERTISERS

DECREASED SEARCH RESULT RANKING

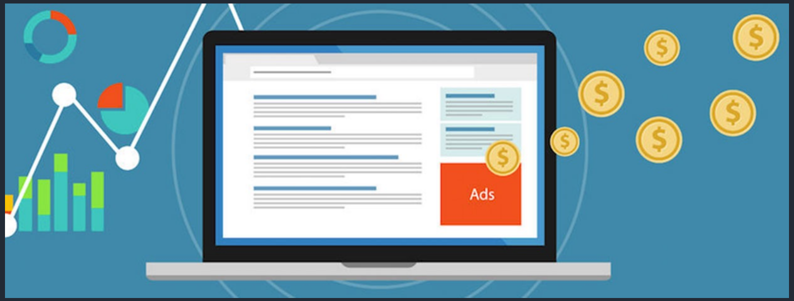
Task: **predicting if an IP address is generating fraudulent clicks**.

It is challenging since the data is massive and skewed towards non fraudulent IP addresses.

kaggle

+

TalkingData



DATASET

Dataset is from a big data service platform(TalkingData) from China that has tracked these frauds by recording the number of clicks, click time and their IP Addresses over an app and checking if they end up installing the advertised application.

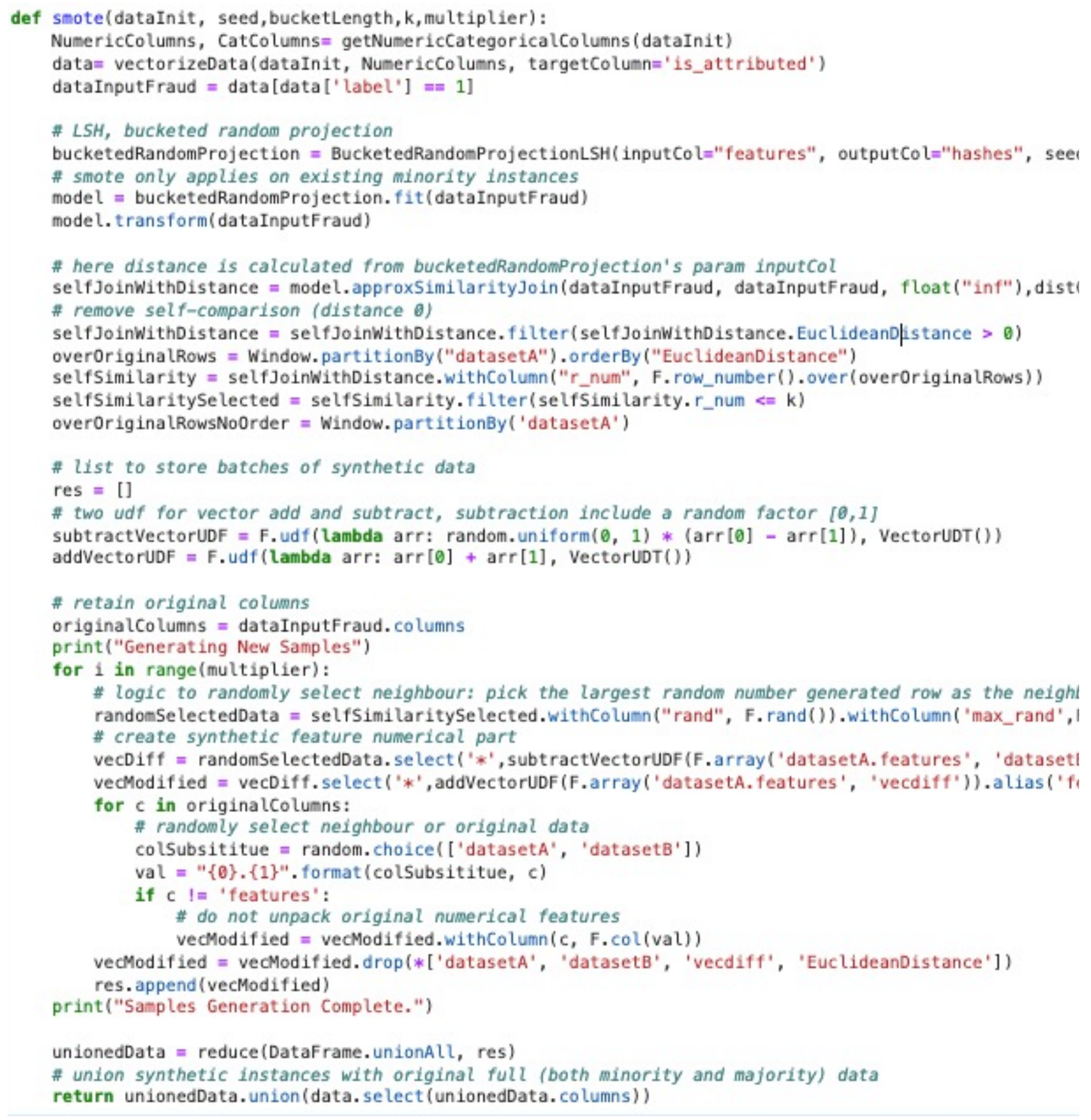
The dataset (train.csv) is a huge (7.54GB) file with more than 1900 Million records having just 4 Million fraudulent labels.

Description of each column:

- **ip**: ip address of click.
- **app**: app id for marketing.
- **device**: device type id of user cell phone (e.g., iphone 14 plus, iphone 13, huawei mate 7, etc.)
- **os**: os version id of user cell phone
- **channel**: channel id of mobile ad publisher
- **click_time**: timestamp of click (UTC)
- **attributed_time**: if user download the app for after clicking an ad, this is the time of the app download
- **is_attributed**: the target that is to be predicted, indicating the app was downloaded

SAMPLING TECHNIQUES

- **Random Under Sampling** means to randomly delete examples in the majority class.
- **Stratified Sampling** is a method for sampling from a population whereby the population is divided into subgroups and units are randomly selected from the subgroups.
- **Random Over Sampling** means to randomly duplicate examples in the minority class.
- **Synthetic Minority Oversampling Technique (SMOTE)** is a technique to up-sample the minority classes while avoiding overfitting. It does this by generating new synthetic examples close to the other points (belonging to the minority class) in feature space. We wrote our custom logic to find out the nearest points and generate synthetic examples.



```python
def smote(dataInit, seed,bucketLength,k,multiplier):
    NumericColumns, CatColumns= getNumericCategoricalColumns(dataInit)
    data= vectorizeData(dataInit, NumericColumns, targetColumn='is_attributed')
    dataInputFraud = data[data['label'] == 1]

    # LSH, bucketed random projection
    bucketedRandomProjection = BucketedRandomProjectionLSH(inputCol="features", outputCol="hashes", see
    # smote only applies on existing minority instances
    model = bucketedRandomProjection.fit(dataInputFraud)
    model.transform(dataInputFraud)

    # here distance is calculated from bucketedRandomProjection's param inputCol
    selfJoinWithDistance = model.approxSimilarityJoin(dataInputFraud, dataInputFraud, float("inf"),dist
    # remove self-comparison (distance 0)
    selfJoinWithDistance = selfJoinWithDistance.filter(selfJoinWithDistance.EuclideanDistance > 0)
    overOriginalRows = Window.partitionBy("datasetA").orderBy("EuclideanDistance")
    selfSimilarity = selfJoinWithDistance.withColumn("r_num", F.row_number().over(overOriginalRows))
    selfSimilaritySelected = selfSimilarity.filter(selfSimilarity.r_num <= k)
    overOriginalRowsNoOrder = Window.partitionBy('datasetA')

    # list to store batches of synthetic data
    res = []
    # two udf for vector add and subtract, subtraction include a random factor [0,1]
    subtractVectorUDF = F.udf(lambda arr: random.uniform(0, 1) * (arr[0] - arr[1]), VectorUDT())
    addVectorUDF = F.udf(lambda arr: arr[0] + arr[1], VectorUDT())

    # retain original columns
    originalColumns = dataInputFraud.columns
    print("Generating New Samples")
    for i in range(multiplier):
        # logic to randomly select neighbour: pick the largest random number generated row as the neigh
        randomSelectedData = selfSimilaritySelected.withColumn("rand", F.rand()).withColumn('max_rand',
        # create synthetic feature numerical part
        vecDiff = randomSelectedData.select('*',subtractVectorUDF(F.array('datasetA.features', 'dataset
        vecModified = vecDiff.select('*',addVectorUDF(F.array('datasetA.features', 'vecdiff')).alias('f
        for c in originalColumns:
            # randomly select neighbour or original data
            colSubsititue = random.choice(['datasetA', 'datasetB'])
            val = "{0}.{1}".format(colSubsititue, c)
            if c != 'features':
                # do not unpack original numerical features
                vecModified = vecModified.withColumn(c, F.col(val))
        vecModified = vecModified.drop(*['datasetA', 'datasetB', 'vecdiff', 'EuclideanDistance'])
        res.append(vecModified)
    print("Samples Generation Complete.")

    unionedData = reduce(DataFrame.unionAll, res)
    # union synthetic instances with original full (both minority and majority) data
    return unionedData.union(data.select(unionedData.columns))
```

DATA SAMPLING PIPELINE | PREPROCESSING

Preprocessing

Count of labels before preprocessing

Labels	Count
Fraud	456846
Not Fraud	184447044
Ratio	**403**

IP	APP	DEVICE	OS	CHANNEL	LABEL
277396	706	3475	800	202	2

Unique Values in each columns.

Count of labels after preprocessing

Labels	Count
Fraud	423712
Not Fraud	53602189
Ratio	**126**

LOW IMBALANCED DATASETS

Stratified Sampling

Under Sampling

Dataset 6M

Labels	Count
Fraud	169973
Not Fraud	446940
Ratio	**4**

Dataset 26M

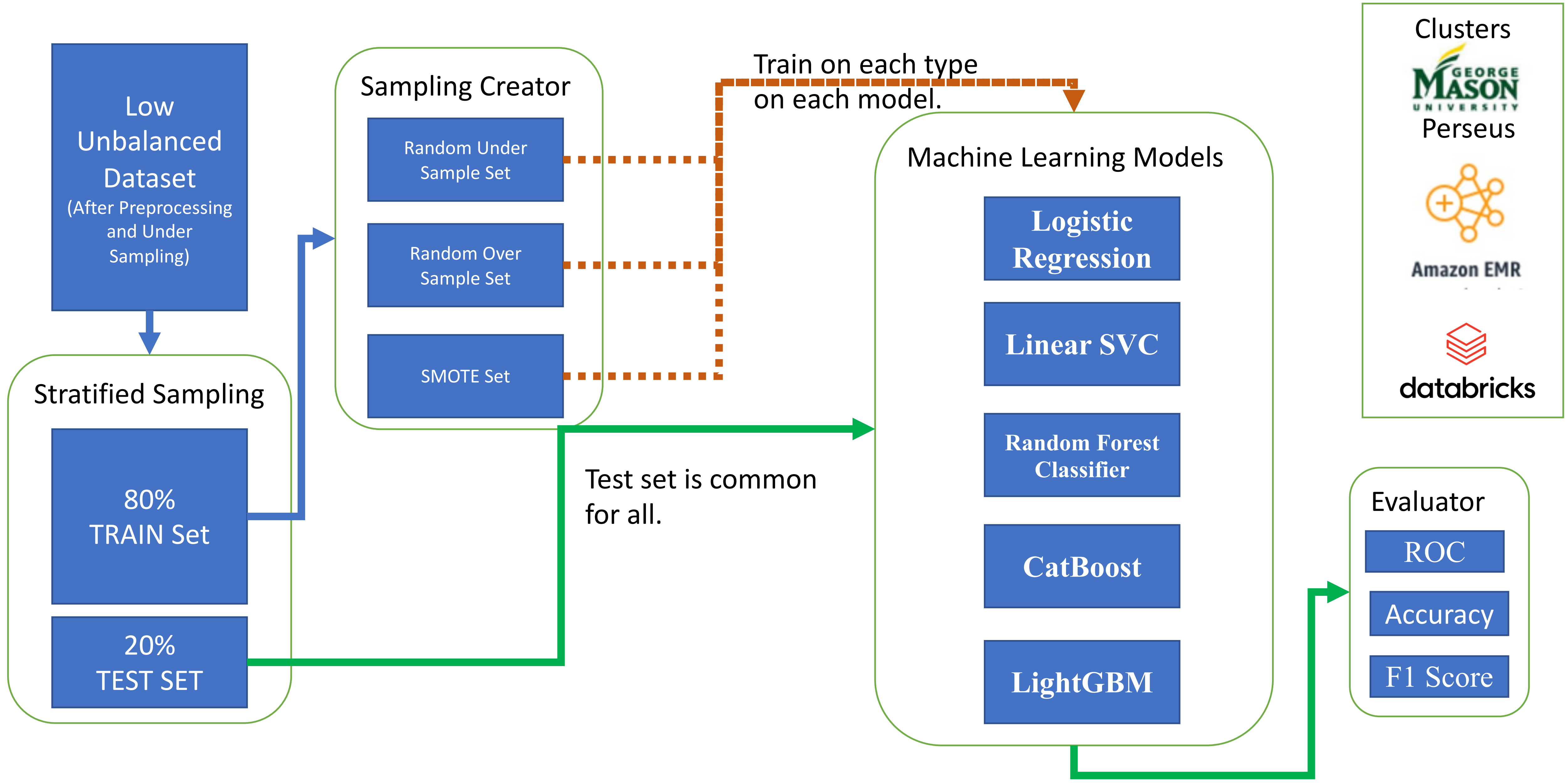
Labels	Count
Fraud	423712
Not Fraud	2147250
Ratio	**5**

Count of labels after down Sampling Not Fraud

Datasets to compare different Sampling techniques

New Datasets using 6M LOW IMBALANCED SET	**New Datasets using 26M LOW IMBALANCED SET**
Under Sampling 6M	Under Sampling 26M
Over Sampling 6M	Over Sampling 26M
SMOTE 6M	SMOTE 26M

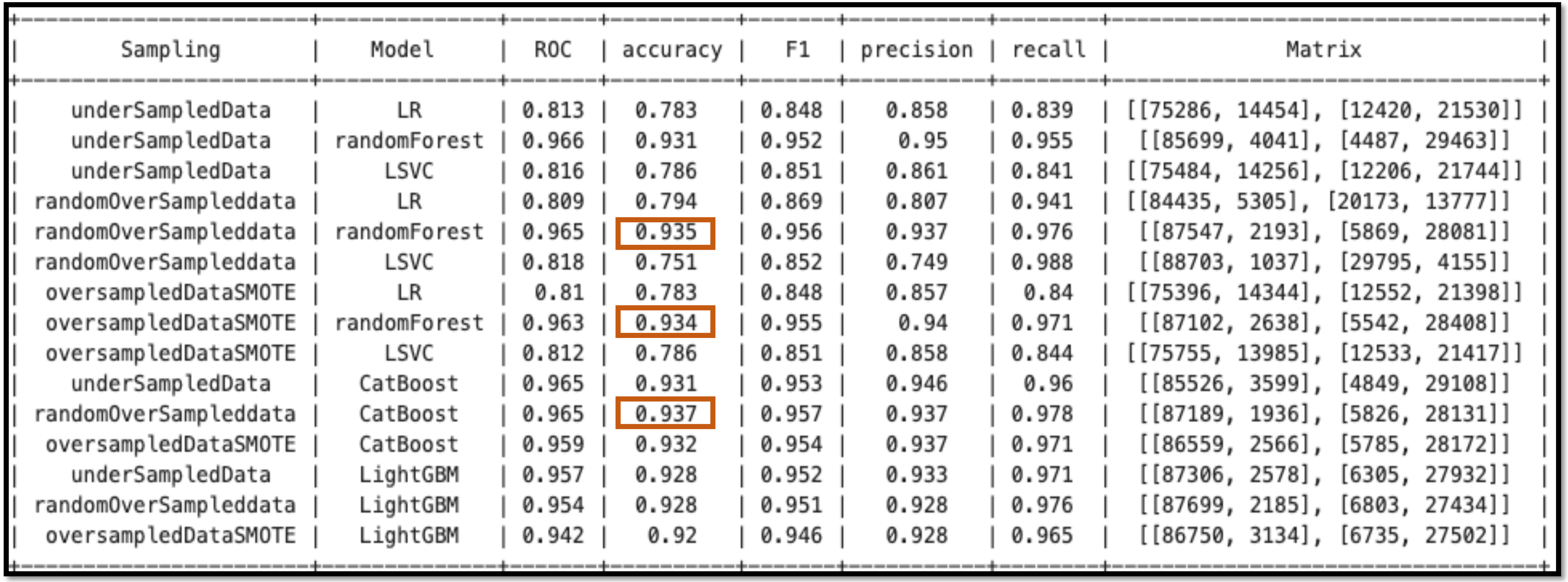
Machine Learning Pipeline - Our Approach



LET’S SEE THE DEMO

RESULTS

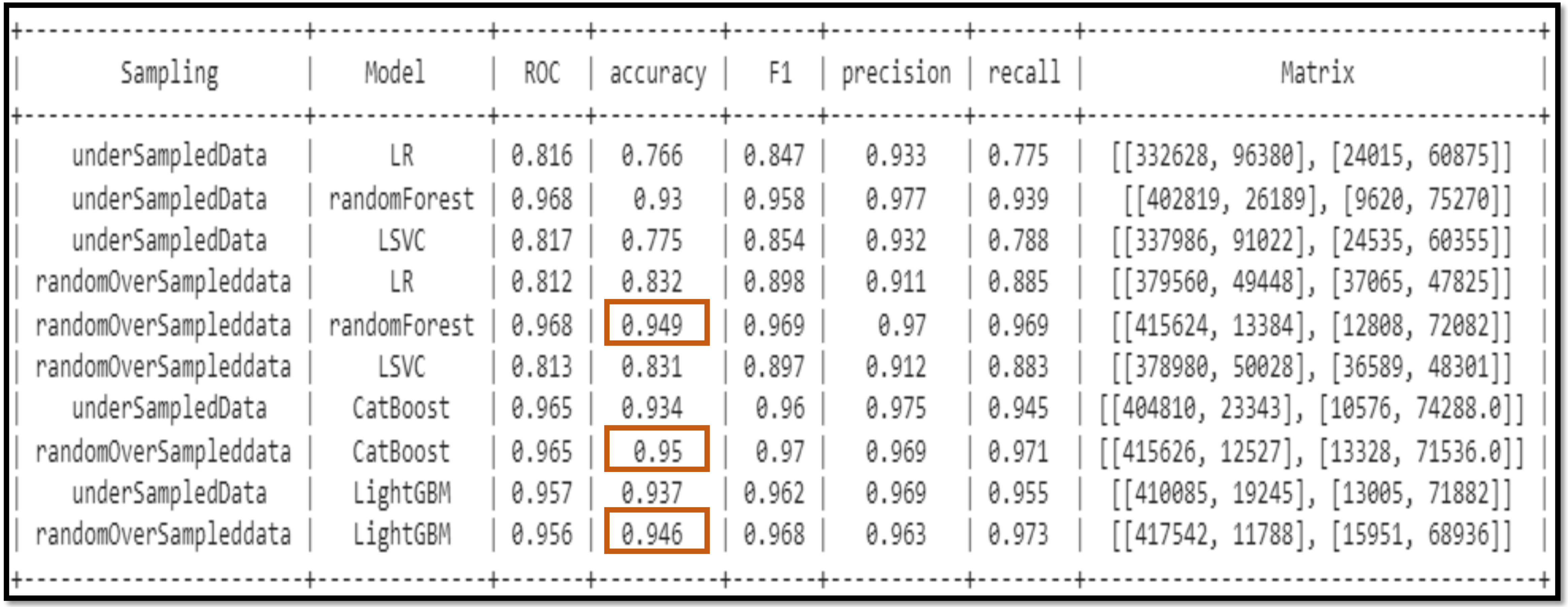
6M Dataset



```
+-----------------------+--------------+-------+----------+-------+-----------+--------+----------------------------------+
|        Sampling       |    Model     |  ROC  | accuracy |   F1  | precision | recall |              Matrix              |
+-----------------------+--------------+-------+----------+-------+-----------+--------+----------------------------------+
|     underSampledData  |      LR      | 0.813 |  0.783   | 0.848 |   0.858   | 0.839  | [[75286, 14454], [12420, 21530]] |
|     underSampledData  | randomForest | 0.966 |  0.931   | 0.952 |    0.95   | 0.955  |  [[85699, 4041], [4487, 29463]]  |
|     underSampledData  |     LSVC     | 0.816 |  0.786   | 0.851 |   0.861   | 0.841  | [[75484, 14256], [12206, 21744]] |
| randomOverSampleddata |      LR      | 0.809 |  0.794   | 0.869 |   0.807   | 0.941  | [[84435, 5305], [20173, 13777]]  |
| randomOverSampleddata | randomForest | 0.965 |  0.935   | 0.956 |   0.937   | 0.976  |  [[87547, 2193], [5869, 28081]]  |
| randomOverSampleddata |     LSVC     | 0.818 |  0.751   | 0.852 |   0.749   | 0.988  |  [[88703, 1037], [29795, 4155]]  |
|  oversampledDataSMOTE |      LR      |  0.81 |  0.783   | 0.848 |   0.857   |  0.84  | [[75396, 14344], [12552, 21398]] |
|  oversampledDataSMOTE | randomForest | 0.963 |  0.934   | 0.955 |    0.94   | 0.971  |  [[87102, 2638], [5542, 28408]]  |
|  oversampledDataSMOTE |     LSVC     | 0.812 |  0.786   | 0.851 |   0.858   | 0.844  | [[75755, 13985], [12533, 21417]] |
|     underSampledData  |   CatBoost   | 0.965 |  0.931   | 0.953 |   0.946   |  0.96  |  [[85526, 3599], [4849, 29108]]  |
| randomOverSampleddata |   CatBoost   | 0.965 |  0.937   | 0.957 |   0.937   | 0.978  |  [[87189, 1936], [5826, 28131]]  |
|  oversampledDataSMOTE |   CatBoost   | 0.959 |  0.932   | 0.954 |   0.937   | 0.971  |  [[86559, 2566], [5785, 28172]]  |
|     underSampledData  |   LightGBM   | 0.957 |  0.928   | 0.952 |   0.933   | 0.971  |  [[87306, 2578], [6305, 27932]]  |
| randomOverSampleddata |   LightGBM   | 0.954 |  0.928   | 0.951 |   0.928   | 0.976  |  [[87699, 2185], [6803, 27434]]  |
|  oversampledDataSMOTE |   LightGBM   | 0.942 |   0.92   | 0.946 |   0.928   | 0.965  |  [[86750, 3134], [6735, 27502]]  |
+-----------------------+--------------+-------+----------+-------+-----------+--------+----------------------------------+
```

RESULTS

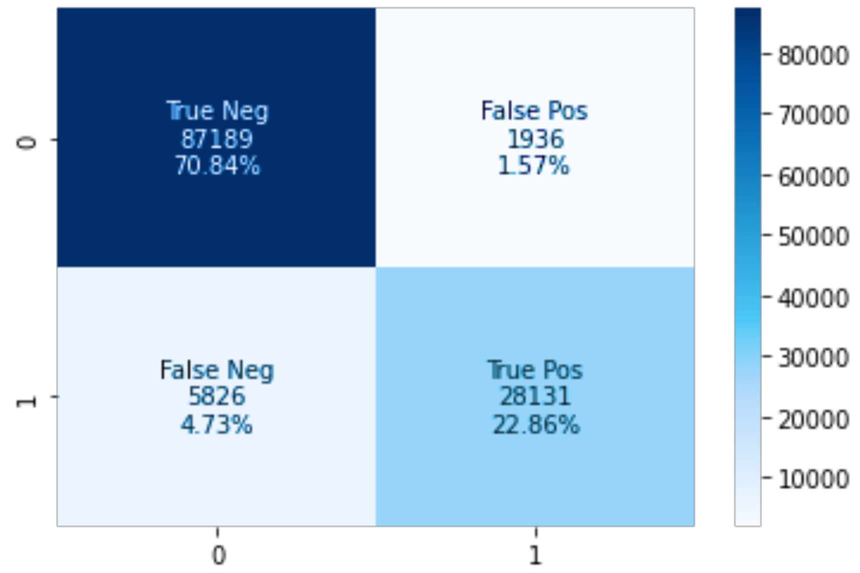
26M Dataset



```
+-----------------------+-------------+-------+----------+-------+-----------+--------+-------------------------------------+
|        Sampling       |    Model    |  ROC  | accuracy |   F1  | precision | recall |                Matrix               |
+-----------------------+-------------+-------+----------+-------+-----------+--------+-------------------------------------+
|    underSampledData   |      LR     | 0.816 |  0.766   | 0.847 |   0.933   | 0.775  |  [[332628, 96380], [24015, 60875]]  |
|    underSampledData   | randomForest| 0.968 |   0.93   | 0.958 |   0.977   | 0.939  |   [[402819, 26189], [9620, 75270]]  |
|    underSampledData   |     LSVC    | 0.817 |  0.775   | 0.854 |   0.932   | 0.788  |  [[337986, 91022], [24535, 60355]]  |
|  randomOverSampleddata|      LR     | 0.812 |  0.832   | 0.898 |   0.911   | 0.885  |  [[379560, 49448], [37065, 47825]]  |
|  randomOverSampleddata| randomForest| 0.968 |  0.949   | 0.969 |    0.97   | 0.969  |  [[415624, 13384], [12808, 72082]]  |
|  randomOverSampleddata|     LSVC    | 0.813 |  0.831   | 0.897 |   0.912   | 0.883  |  [[378980, 50028], [36589, 48301]]  |
|    underSampledData   |   CatBoost  | 0.965 |  0.934   |  0.96 |   0.975   | 0.945  | [[404810, 23343], [10576, 74288.0]] |
|  randomOverSampleddata|   CatBoost  | 0.965 |   0.95   |  0.97 |   0.969   | 0.971  | [[415626, 12527], [13328, 71536.0]] |
|    underSampledData   |   LightGBM  | 0.957 |  0.937   | 0.962 |   0.969   | 0.955  |  [[410085, 19245], [13005, 71882]]  |
|  randomOverSampleddata|   LightGBM  | 0.956 |  0.946   | 0.968 |   0.963   | 0.973  |  [[417542, 11788], [15951, 68936]]  |
+-----------------------+-------------+-------+----------+-------+-----------+--------+-------------------------------------+
```

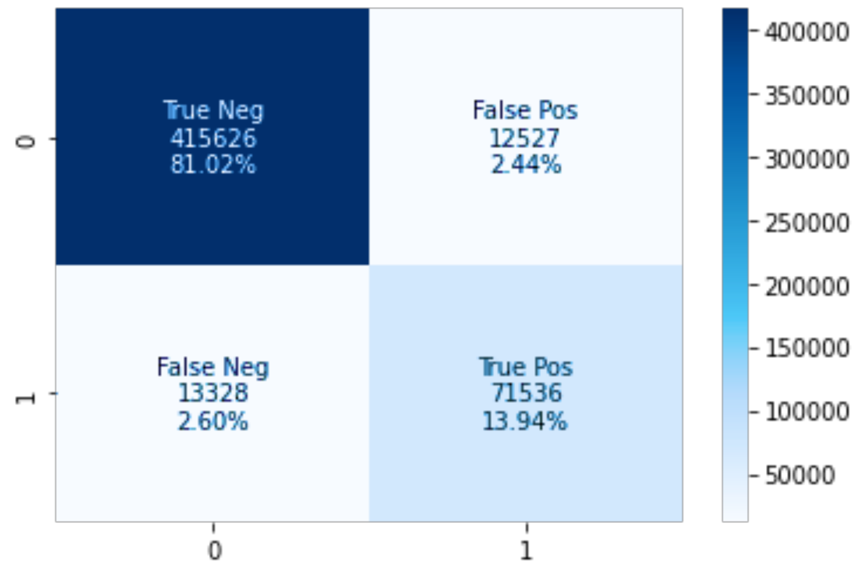
Confusion Matrix of Best Model

6M Dataset



Confusion Matrix for CatBoost with Random Oversampling as per Accuracy = 93.7%

26M Dataset



Confusion Matrix for CatBoost with Random Oversampling as per Accuracy = 95%

LEARNINGS | CONCLUSIONS | FUTURE WORK

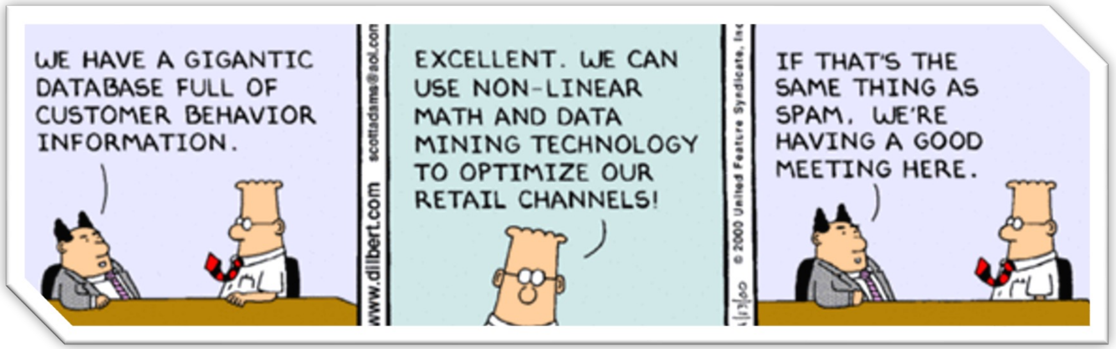
NEW LEARNINGS:

- Comparing and combining different Sampling methods.
- Implementing algorithms such as SMOTE which are not present in the spark library.
- Integrating Machine learning models originally written in Java with PySpark by adding external Jar files and python wrappers.
- Configuring and using AWS ElasticMapReduce clusters for machine learning tasks with high compute requirements.

CONCLUSIONS: Using under sampling to convert highly imbalanced data to low imbalanced data and then applying oversampling to give balanced data gives us the best result of **accuracy of 93.7% on 6M Dataset and 95% on 26M Dataset**.

FUTURE WORK: Try to use predictions from all the models and apply ensemble technique. Each of the 5 different model's predictions column could be merged, and majority voting to be used to predict the label.

In case of questions:
Please reach us out at:
1. Janit Bidhan jbidhan@gmu.edu
2. Sreenivasa Rayaprolu: srayapr@gmu.edu


WE HAVE A GIGANTIC DATABASE FULL OF CUSTOMER BEHAVIOR INFORMATION.
EXCELLENT. WE CAN USE NON-LINEAR MATH AND DATA MINING TECHNOLOGY TO OPTIMIZE OUR RETAIL CHANNELS!
IF THAT'S THE SAME THING AS SPAM, WE'RE HAVING A GOOD MEETING HERE.
www.dilbert.com
scottadams@aol.com
© 2000 United Feature Syndicate, Inc.