ERROR 404

ECS 171: ML FINAL PROJECT

Jack Chervet, Sagar Gupta, Janet Loyola, Aria Chang

What is given and what are we supposed to do?

This data corresponds to a set of financial transactions associated with individuals. For each observation, it was recorded whether a default was triggered. In case of a default, the loss was measured. This quantity lies between 0 and 100. It has been normalized. For example, a loss of 60 means that only 40 is reimbursed. If the loan did not default, the loss was 0. We are given a training data set of 50,000 rows and 769 features, and the last column is the loss. We are also given a testing data set and we are supposed to predict whether or not the loan will default or not based on the algorithms that we rand on our training data.

GOAL: We try to predict the losses for each observation in the test set.

How did we clean the dataset and reduce the dimensions?

We took the following steps to reduce the dimensions of the dataset.

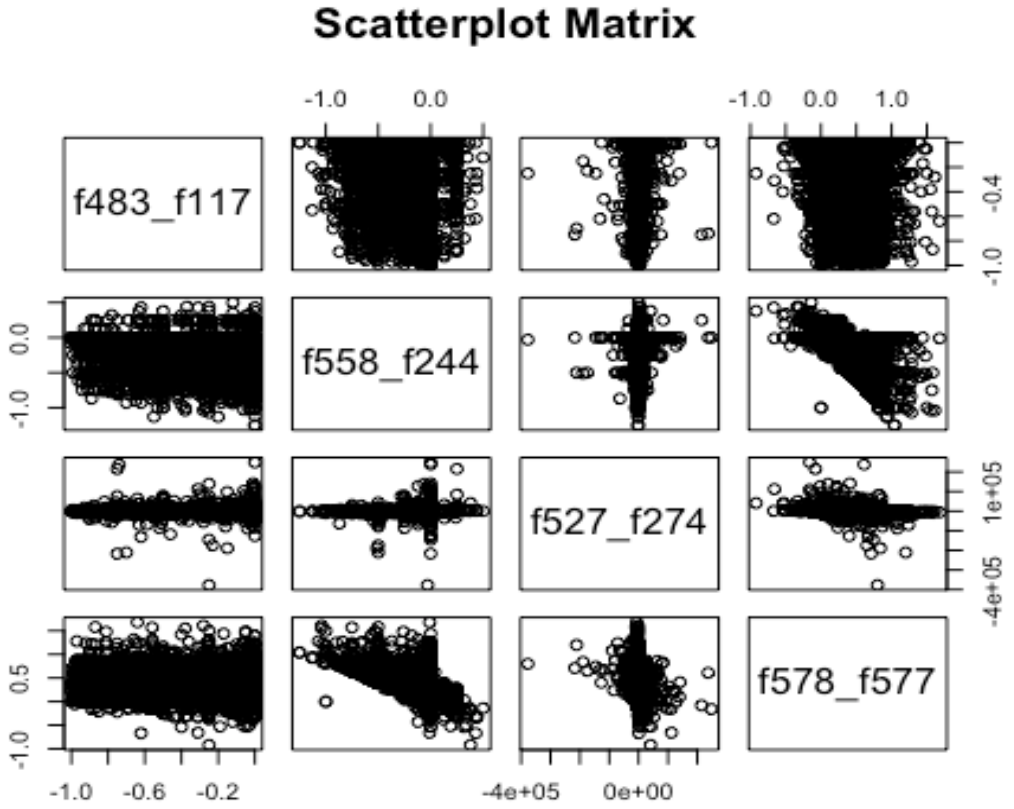
- Initially, we converted all the NA values to 0.
- Then, we created a matrix of the correlation of every feature with every other.
- Then, we took the feature pairs that are highly correlated (correlation between 0.9999 and 1) and returned those feature pairs.

```
def correlation(dataset, threshold):
    col_corr = set() # Set of all the names of deleted columns
    corr_matrix = dataset.corr() # create correlation matrix

    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if corr_matrix.iloc[i, j] >= threshold and corr_matrix.iloc[i, j] < 1: # if value above our threshold
                colname = corr_matrix.columns[i] # getting the name of column
                col_corr.add( (colname, corr_matrix.columns[j]) ) # add column pairs to set
                if colname in dataset.columns:
                    del dataset[colname] # deleting the column from the dataset

    return col_corr #set of all the pairs of features with very high correlation
```

- Then, we took the pairs of features with very high correlation (about 200 pairs), and created new features by taking the difference between the pairs of highly correlated features and hence obtained around 200 new features. **This was done in an attempt to solve the issue of over fitting due to the noise in the original features.**



- From this scatterplot, we see that these four features are not correlated. Similarly, we checked several other features to check the correlations (THIS STEP IS JUST TO CONFIRM NO CORRELATIONS).
- Then, we used Principal Component Analysis to get 35 new features based on the combination of the old features. So what does Principal Component Analysis (PCA) do? I will talk about this in the next section.
- Now, we cleaned the data with about 200 new correlated features based on the difference and 35 features based on PCA. Hence, we have reduced the features from 769 to around 235. Now, the data is ready to help us predict whether or not the loan will default.

So, what is Principal Component Analysis and why did we do this?

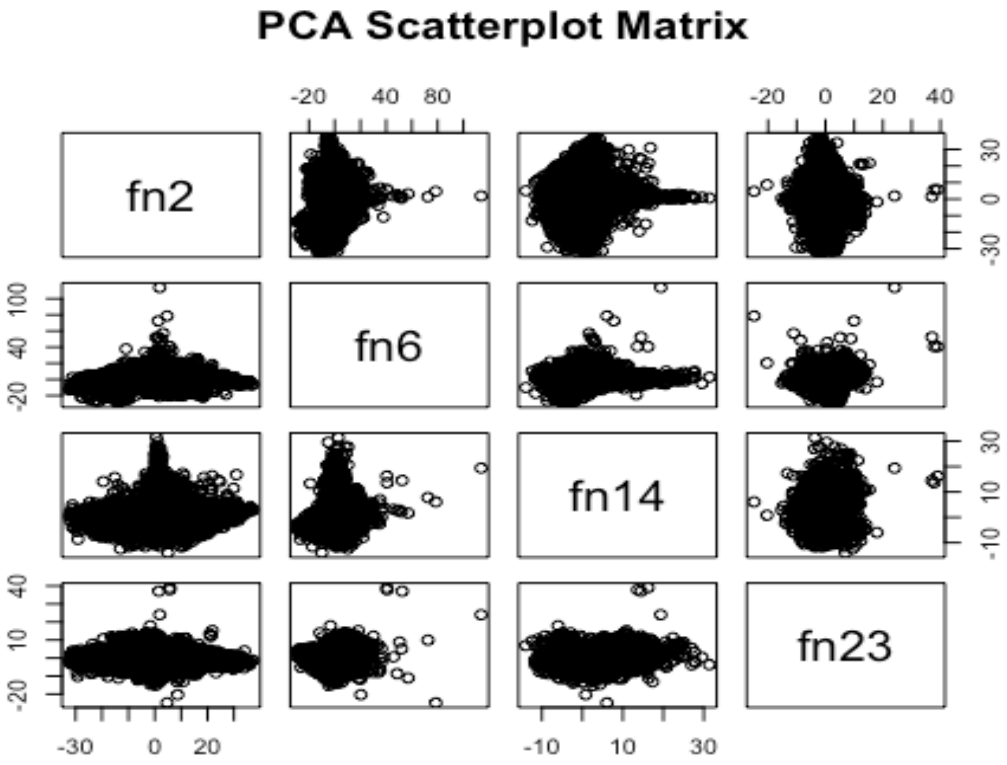
We wanted to do some feature reduction with feature extraction, and hence wanted to form new variables from the old variables based on maximum variability. This is why we decided to do Principal Component Analysis. Using PCA, we reduced the data from the original 769 features to 35 features based on the linear combination of uncorrelated variables.

What are the steps of PCA?

- First, we needed to normalize the data. Why? Normalization makes sure that your end result is not dominated by a single variable. So we have to standardize the data set, as different variables are measured in different scales.
- Then, we used the inbuilt PCA function to get the respective 35 components.

```
traindf = StandardScaler().fit_transform(traindf) # Normalize the data
pca = PCA(n_components=35) # Apply PCA to reduce the original 769 vectors to 35 components
'''*Note: We chose the number 35 after trial and error as it seemed to consistly provide the best results for our model'''
principalComponents = pca.fit_transform(traindf) # Fit the data based on the PCA model
```

From the below scatterplot, we see that these four features are not correlated. Similarly, we checked several other features from PCA to check the correlations (THIS STEP IS JUST TO CONFIRM NO CORRELATIONS).

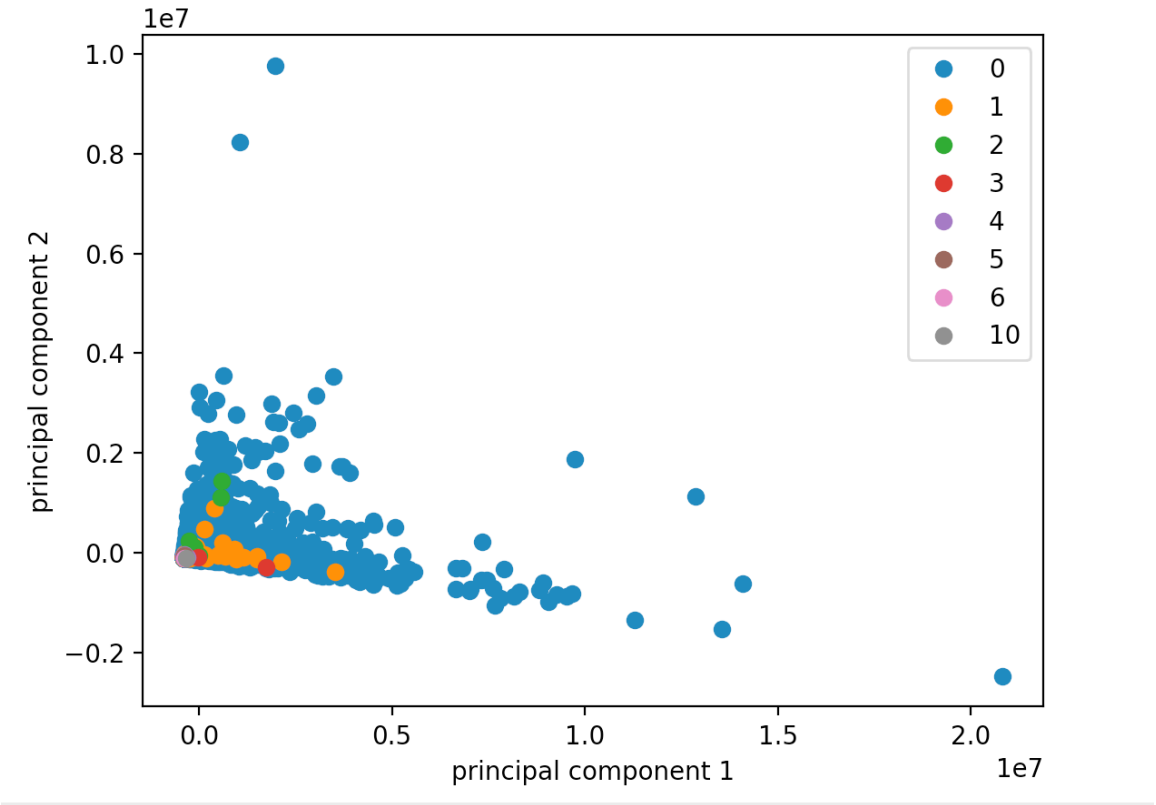


- But what exactly does this PCA function do in the background?
  - Transform our features data frame into a matrix
  - Calculate covariance on feature matrix
  - Find eigenvalues and eigenvectors. The eigenvalues tells the variance in the data set. Eigenvectors tell us the corresponding direction of the variance.
  - Selects the top 35 eigenvalues and corresponding eigenvectors.

- Finally, it forms the new dataset in reduced dimensions.

However, our team wanted to visualize the data set to get an idea of it. However, because the data had a lot of features, it was impossible to visualize such high dimensional data. Hence, in the below plot I only used a subset of the original data with 4000 rows and 10 new features (combination of the old features).
Both Principal Component 1 and Principal Component 2 are made up of linear combinations of your features. They are also orthogonal of each other and thus the components are independent. **PCA is used as a method to reduce multi-collinearity in the data and reduce the dimensions of your data.** If 95% of the data set is explained in the first two principal components it is easy to visualize clusters based on the Principal Components. But in order to interpret what they mean we looked at the Principal Components and see which features are largest in that Principal Component. However, by changing that underlined part in the code from 10 to something else, we get a different plot every time as we are changing the number of categories we want.

```
final_df['label_round'] = final_df['label'].map(lambda x: int(x/10))
```



Now, we take a look at the scatterplot of a few features from the PCA just to confirm that there are no correlations between the features.

Few definitions of the models and tools that come up in the next section

- **SelectKBest:**
  SelectKBest selects the top k features that have maximum relevance with the target variable (in this case the loss). It takes two parameters as input arguments, "k" and the score function to rate the relevance of every feature with the loss.

- **Stochastic Gradient Descent Classifier:**
  Stochastic Gradient Descent (SGD) is a technique that is used to find the minima of the function. Stochastic Gradient Descent Classifier is a linear classifier that uses SGD for training. It basically looks for the minima of the loss using SGD.

- **Gradient Boosting Regressor:**
  Gradient boosting regressors are a type of inductively generated tree ensemble model. At each step, a new tree is trained against the negative gradient of the loss function, which is analogous to (or identical to, in the case of least-squares error) the residual error.

How did we end up selecting the models and algorithms to use?

For our model, we tried several different approaches.

• First, we cleaned the data using scikit-learn method called **SelectKBest**. We then binarized the loss, and attempted to first use a model for only classification to see if we could beat the accuracy score and MAE of predicting all 0s. For this model, we used **Stochastic Gradient Descent on logistic regression** (SGDClassifer from scikit-learn). Unfortunately, we were not able to beat the scores achieved when simply predicting all zeros.
-Since only classification was not working, we decided to try to use a regression model to predict all loss, ranging from 0 to 100. We again used scikit-learn's SelectKBest function to clean our data, and we scored a Mean Absolute Error (MAE) of 0.8007 with this model. This seemed great, but upon inspection of the predictions we noticed that the model was predicting a loss of 0 for every example. When applied to the test data set (which has more features), the MAE would come out to be 0.83414 as shown on Kaggle leaderboards when submitting a prediction of all 0.
-We believe that as about 90% of the time there is no default, the 10% of times when there is a default are massively drowned out by the times when there is no default and so the model predicts a 0 every time.

• Since neither of these models were working alone, we figured we may be able to improve our score by using them together. To do this, we used SelectKBest to clean the data, and then binarized the loss and trained SGDClassifier to predict when the

examples would default. We also selected out the examples that had a loss in the training data, and used only those examples (4626 of the 50000 total examples) to train our GradientBoostingRegressor to learn how much of a loss should be predicted in the event of a default. Using these two models together, we were able to score accuracies of around 90-91% on classification (similar to a prediction of all zeros), but unfortunately could not get an MAE lower than .84 on the training data.

• At this point, we realized that we would have to clean the data differently and select better features. Hence, we decided to use PCA to come up with 35 new features based on the original 769. We also counted the correlation between every pair of features, and selected features that had a correlation greater than .9999 but less than 1. This yielded around 200 pairs of features. We took the difference of each pair of features to create 200 new features and concatenated these with the 35 PCA features we had previously generated.

• Finally, we used this new cleaned up data with our model that employed both Stochastic Gradient Descent Classifier and Gradient Boosting Regressor. This improved our accuracy and gave a less Mean Absolute Error (around 0.7).

```
clf = linear_model.SGDClassifier(loss='log', max_iter=800) #define classification model
scores = cross_val_predict(clf, principalDf, loss, cv=5) #do a 5 fold cross validation on th data set using the model
print("Accuracy of model: %3f" % (accuracy_score(loss, scores))) #Checking accuracies
print("Accuracy of all 0s: %3f" % (accuracy_score(loss, np.zeros((50000,)))))
```

SGDClassifier

```
# GBT Model with learning rate of 0.1
est = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=1, random_state=0, loss='lad')
regressionScores = cross_val_predict(est, regressionDF, a, cv=3) # 3 fold cross validation

est.fit(regressionDF, a) #fit the data based on the model
```

GradientBoostingRegressor

Conclusion:

In the end, the model with classification and regression in conjunction with the PCA and the 200 created features worked best, and we scored a 0.7 MAE on the Kaggle leaderboards.

CODE APPENDIX:

```
import pandas as pd
import numpy as np
import random
import os
import sys
import math
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import f_regression
from sklearn.model_selection import cross_val_predict
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
from sklearn import linear_model
from sklearn import svm
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

def parseTrainData():
    trainData = np.load('./ecs171train.npy')
    df = pd.DataFrame([sub.decode('UTF-8').split(",") for sub in
trainData[1:]])
    df.columns = trainData[0].decode('UTF-8').split(',')
    df.set_index('id', inplace=True)
    df = df.apply(pd.to_numeric, errors='coerce')
    df = df.fillna(value=0)
    df.to_csv("trainData.csv", columns = trainData[0].decode('UTF-
8').split(','))

    testData = np.load('./ecs171test.npy')
    tdf = pd.DataFrame([sub.decode('UTF-8').split(",") for sub in testData])
    headers = trainData[0].decode('UTF-8').split(',')
    tdf.columns = headers[:(len(headers)-1)]
    tdf.set_index('id', inplace=True)
    tdf = tdf.apply(pd.to_numeric, errors='coerce')
    tdf = tdf.fillna(value=0)
    tdf.to_csv("testData.csv", columns = trainData[0].decode('UTF-
8').split(','))
```

```
def makeFeatures(df):
    flist = {('f770', 'f52'), ('f553', 'f532'), ('f483', 'f117'), ('f633',
'f408'), ('f729', 'f644'), ('f722', 'f408'), ('f770', 'f345'), ('f559',
'f246'), ('f729', 'f408'), ('f644', 'f74'), ('f644', 'f354'), ('f417', 'f42'),
('f558', 'f244'), ('f527', 'f274'), ('f608', 'f7'), ('f528', 'f527'), ('f608',
'f478'), ('f559', 'f245'), ('f633', 'f379'), ('f770', 'f42'), ('f504', 'f503'),
('f345', 'f58'), ('f553', 'f8'), ('f569', 'f568'), ('f722', 'f633'), ('f74',
'f36'), ('f354', 'f36'), ('f379', 'f52'), ('f608', 'f488'), ('f767', 'f405'),
('f559', 'f558'), ('f633', 'f427'), ('f494', 'f98'), ('f568', 'f255'), ('f559',
'f247'), ('f770', 'f362'), ('f362', 'f58'), ('f504', 'f106'), ('f633', 'f36'),
('f582', 'f8'), ('f493', 'f98'), ('f484', 'f118'), ('f417', 'f58'), ('f417',
'f36'), ('f577', 'f267'), ('f644', 'f371'), ('f427', 'f58'), ('f371', 'f58'),
('f452', 'f85'), ('f548', 'f234'), ('f770', 'f644'), ('f493', 'f95'), ('f408',
'f74'), ('f408', 'f354'), ('f539', 'f225'), ('f494', 'f96'), ('f729', 'f52'),
('f58', 'f52'), ('f578', 'f577'), ('f484', 'f483'), ('f354', 'f52'), ('f74',
'f52'), ('f379', 'f36'), ('f543', 'f8'), ('f569', 'f255'), ('f453', 'f452'),
('f770', 'f633'), ('f371', 'f52'), ('f770', 'f36'), ('f578', 'f267'), ('f569',
'f257'), ('f722', 'f42'), ('f453', 'f87'), ('f644', 'f417'), ('f345', 'f36'),
('f427', 'f42'), ('f568', 'f257'), ('f417', 'f408'), ('f741', 'f633'), ('f722',
'f36'), ('f493', 'f96'), ('f452', 'f88'), ('f569', 'f254'), ('f345', 'f42'),
('f427', 'f36'), ('f504', 'f108'), ('f362', 'f52'), ('f539', 'f224'), ('f568',
'f254'), ('f483', 'f118'), ('f453', 'f85'), ('f741', 'f644'), ('f578', 'f266'),
('f558', 'f246'), ('f722', 'f644'), ('f644', 'f36'), ('f770', 'f722'), ('f549',
'f548'), ('f528', 'f274'), ('f379', 'f58'), ('f453', 'f88'), ('f768', 'f406'),
('f549', 'f235'), ('f539', 'f227'), ('f452', 'f87'), ('f578', 'f265'), ('f770',
'f417'), ('f549', 'f236'), ('f408', 'f345'), ('f729', 'f58'), ('f494', 'f493'),
('f504', 'f107'), ('f503', 'f108'), ('f379', 'f42'), ('f770', 'f741'), ('f408',
'f52'), ('f532', 'f8'), ('f494', 'f95'), ('f427', 'f408'), ('f503', 'f105'),
('f408', 'f42'), ('f417', 'f52'), ('f484', 'f115'), ('f577', 'f264'), ('f741',
'f42'), ('f608', 'f467'), ('f362', 'f36'), ('f484', 'f116'), ('f538', 'f226'),
('f770', 'f371'), ('f543', 'f532'), ('f408', 'f371'), ('f577', 'f266'),
('f633', 'f417'), ('f770', 'f729'), ('f644', 'f379'), ('f741', 'f52'), ('f408',
'f362'), ('f563', 'f8'), ('f568', 'f256'), ('f484', 'f117'), ('f578', 'f264'),
('f548', 'f237'), ('f644', 'f362'), ('f371', 'f36'), ('f539', 'f538'), ('f608',
'f498'), ('f625', 'f389'), ('f569', 'f256'), ('f74', 'f58'), ('f354', 'f58'),
('f549', 'f237'), ('f453', 'f86'), ('f644', 'f345'), ('f558', 'f247'), ('f741',
'f36'), ('f644', 'f52'), ('f371', 'f42'), ('f563', 'f532'), ('f729', 'f42'),
('f633', 'f371'), ('f722', 'f58'), ('f644', 'f42'), ('f538', 'f225'), ('f770',
'f58'), ('f74', 'f42'), ('f354', 'f42'), ('f427', 'f52'), ('f549', 'f234'),
('f770', 'f408'), ('f494', 'f97'), ('f770', 'f74'), ('f539', 'f226'), ('f770',
'f354'), ('f408', 'f58'), ('f452', 'f86'), ('f483', 'f115'), ('f722', 'f52'),
('f608', 'f439'), ('f573', 'f8'), ('f538', 'f227'), ('f559', 'f244'), ('f608',
```

```
'f457'), ('f483', 'f116'), ('f644', 'f58'), ('f362', 'f42'), ('f770', 'f379'),
('f582', 'f532'), ('f644', 'f427'), ('f408', 'f379'), ('f729', 'f633'),
('f503', 'f106'), ('f573', 'f532'), ('f633', 'f345'), ('f558', 'f245'),
('f741', 'f58'), ('f503', 'f107'), ('f729', 'f36'), ('f741', 'f408'), ('f548',
'f236'), ('f633', 'f362'), ('f345', 'f52'), ('f52', 'f42'), ('f548', 'f235'),
('f633', 'f74'), ('f504', 'f105'), ('f633', 'f354'), ('f577', 'f265'), ('f608',
'f508'), ('f644', 'f408'), ('f52', 'f36'), ('f58', 'f36'), ('f493', 'f97'),
('f770', 'f427'), ('f538', 'f224'), ('f42', 'f36')}

    newdf = pd.DataFrame()
    for feature in flist:
        name = str(feature[0])+"_"+str(feature[1])
        newdf[name] = df[feature[0]].sub(df[feature[1]], axis=0)

    # print(newdf)

    return newdf

def main():
    # parseTrainData()
    df = pd.read_csv("trainData.csv")
    tdf = pd.read_csv("testData.csv")
    testid = tdf.id
    df.set_index('id', inplace=True)
    tdf.set_index('id', inplace=True)

    #df.columns = trainData[0].decode('UTF-8').split(',') # put back headers
    df = df.fillna(value=0) #fill nans
    df = df.rename(columns={ df.columns[769]: "loss" })
    tdf = tdf.fillna(value=0) #fill nans

    # print(df)
    og_loss = pd.DataFrame(df['loss']) #store loss column
    # print(og_loss)

    print("tdf")
    print(tdf)

    '''''''''''''''
    #binarize the loss
    traindf = pd.DataFrame(df)
    traindf['loss'][traindf['loss'] > 0] = 1
    loss = traindf['loss']
```

```
    traindf = traindf.drop(['loss'], axis =1) #remove loss column for training
    # ''''''''''''''

    testdf = pd.DataFrame(tdf)
    print("testdf")
    print(testdf)

    newFeatures = makeFeatures(traindf)
    newFeatures_test = makeFeatures(testdf)
    print("newFeatures_test")
    print(newFeatures_test)

    traindf = StandardScaler().fit_transform(traindf)
    pca = PCA(n_components=35)
    principalComponents = pca.fit_transform(traindf)

    testdf = StandardScaler().fit_transform(testdf)
    pca_test = PCA(n_components=35)
    principalComponentsTest = pca_test.fit_transform(testdf)

    # print(principalComponents)
    print("principalComponents.shape")
    print(principalComponents.shape)

    flist = ['fn1', 'fn2', 'fn3', 'fn4', 'fn5', 'fn6', 'fn7', 'fn8', 'fn9',
'fn10']
    principalDf1 = pd.DataFrame(data = principalComponents)
    principalDf1_test = pd.DataFrame(data = principalComponentsTest)

    newFeatures.reset_index(drop=True, inplace=True)
    principalDf = pd.concat([principalDf1, newFeatures], axis=1)

    newFeatures_test.reset_index(drop=True, inplace=True)
    principalDf_test = pd.concat([principalDf1_test, newFeatures_test], axis=1)
    print("principalDf_test")
    print(principalDf_test)


    clf = linear_model.SGDClassifier(loss='log', max_iter=800)
    scores = cross_val_predict(clf, principalDf, loss, cv=5)
    print("Accuracy of model: %3f" % (accuracy_score(loss, scores)))
    print("Accuracy of all 0s: %3f" % (accuracy_score(loss,
np.zeros((50000,)))))
```

```
    clf.fit(principalDf, loss)
    prediction = clf.predict(principalDf)
    prediction_test = clf.predict(principalDf_test)

    lossTrainingIndex = []
    for i in range(0, len(loss)):
        if loss.as_matrix()[i] == 1:
            lossTrainingIndex.append(i)

    regressionDF = principalDf.loc[lossTrainingIndex[:]]
    regressionLoss = pd.DataFrame(og_loss)
    regressionLoss = regressionLoss[regressionLoss['loss'] > 0]

    #
    # print(loss)
    # principalDf.append(loss, ignore_index=True)
    # print(principalDf)

    # regressionDF = principalDf[principalDf.loss != 0]
    # print(regressionDF.shape)
    # regressionDF = regressionDF.drop(['loss'], axis =1)

    # og_loss = og_loss.tolist()
    # og_loss = list(filter(lambda a: a != 0, og_loss))

    a = regressionLoss.as_matrix()
    a = a.flatten()

    est = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,
max_depth=1, random_state=0, loss='lad')
    regressionScores = cross_val_predict(est, regressionDF, a, cv=3)
    print("Calculating MAE")
    print("MAE (Actual loss vs regressionScores): %3f" %
(mean_absolute_error(a, regressionScores)))

    est.fit(regressionDF, a)

    lossPredictedIndex = []
    for i in range(0, len(prediction)):
        if prediction[i] ==1:
            lossPredictedIndex.append(i)
```

```
    toPredict_reg = principalDf.loc[lossPredictedIndex[:]]

    prediction_reg = est.predict(toPredict_reg)

    for i in range(0, len(prediction_reg)):
         prediction[lossPredictedIndex[i]] = prediction_reg[i]

    print("Final MAE (prediction vs og_loss): %3f" %
(mean_absolute_error(og_loss, prediction)))
    print("Control MAE (all-0s vs og_loss): %3f" %
(mean_absolute_error(og_loss, np.zeros((50000,)))))

    lossPredictedIndex_test = []
    for i in range(0, len(prediction_test)):
        if prediction_test[i] ==1:
            lossPredictedIndex_test.append(i)

    toPredict_reg_test = principalDf_test.loc[lossPredictedIndex_test[:]]

    prediction_reg_test = est.predict(toPredict_reg_test)

    for i in range(0, len(prediction_reg_test)):
         prediction_test[lossPredictedIndex_test[i]] = prediction_reg_test[i]

    ids = np.array(testid.tolist())
    # output_tuples = np.zeros(((len(ids)+1),2))
    # output_tuples[0][0] = "id"
    # output_tuples[0][1] = "loss"
    # for i in range(1, (len(ids)+1)):
    #     output_tuples[i][0] = ids[i]
    #     output_tuples[i][1] = prediction_test[i]

    output_tuples = np.column_stack((ids, prediction_test))
    output_tuples = output_tuples.astype(str)
    output_tuples = np.insert(output_tuples, 0, ['id', 'loss'], axis=0 )

    np.savetxt("output_tuples.csv", output_tuples, delimiter=",", fmt="%s")

if __name__ == "__main__":
    main()
```