My approach

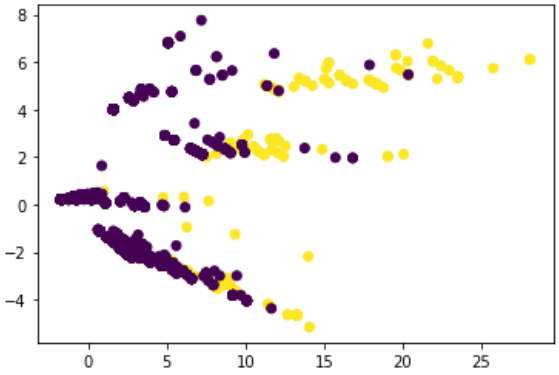
To start with solving this problem my approach formalizes more and more as my analysis of the data increases. The first thing to cross my mind was the imbalance between the positive (match) class and negative(unmatched) class which is 1:12

The Class 0: 858 (7%) and Class 1: 11176 (93%). With this, I make my mind clear that the problem is Binary classification with imbalanced data.

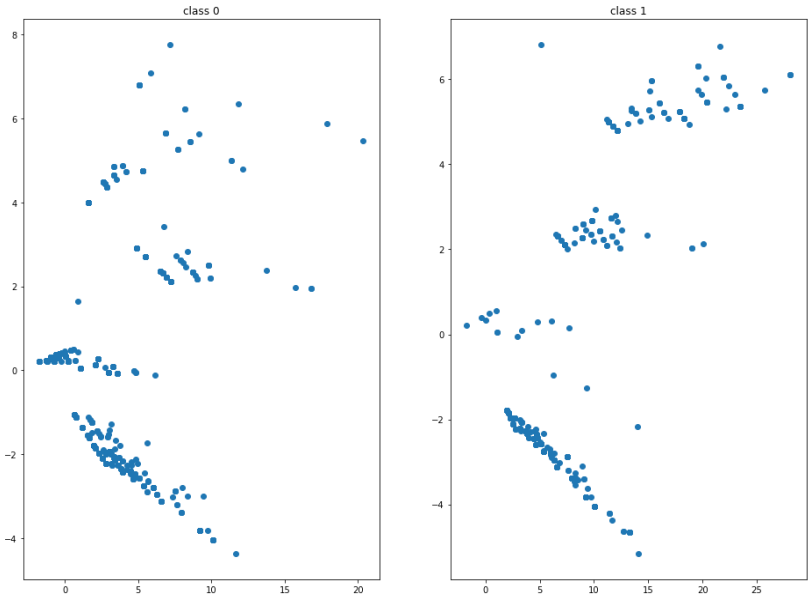
This also leads me to think about doing more extensive feature engineering so I can increase the recall and keep precision high (as usually difficult in these problems). Also, the name or relation b/w the features is really not known to me. so I would like to create some features based on interactions and want to know do those came into play for modeling.

And I also did some small plotting and found that interactive features might be better in predicting as most of the features though not significantly help us in classification but some feature engineering could help us in this

After creating the interaction features with a degree of 3. I did some more analysis by doing PCA and plotting the points in 2D space and found an interesting detail to look at The first image to look at was



This graph is a plot of PCA components where class 0 is Purple and class 1 is yellow From this, we can see that we can separate them not linearly but in higher dimension might be. (one more thing to create more versatile features)
But when I plot the classes side by side I got some interesting information that would like to share. When plotting the classes I saw that a cluster of data points is overlapping



This lead me to think that it might get separated in higher dimensions but with some fast modeling, I found that the metric does not seem to increase so I got stuck. This leads me to create more dynamic features.

I tried to create interaction, groupby, and aggregation-based features on top of sklearn-made polynomial features with degree = 3. All in all, I ended up with 536 features. These many could easily lead to a curse of dimensionality problems + could take too much time to train. So collected 50 PCA components for training with this I used metrics like - F1_score, Accuracy. Also Not used ROCAUC because the false positive rate for highly imbalanced datasets is pulled down due to a large number of true negatives.

The best F1_score for a fold that I have got is 92% macro and 85% on class 1 and 98% accuracy. But the final metrics would lead to an **average of 85-87% f1_score and 96% accuracy.**

Some of my suggestions to the Software Engineers is that they can look at my prediction_pipeline function where they can see a commented line telling the sum of positive classes predicted on different thresholds. The lower the threshold the more model predicts a positive class. So they can play with these parameters and can attenuate and get the exact thresholds according to the need.

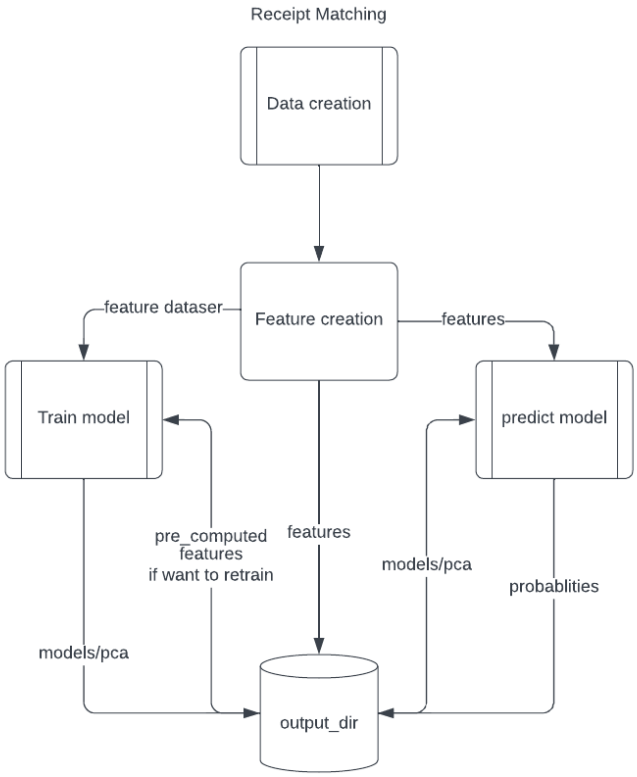
So In a more simple sense, our model identifies all of our positive matches and that is at the same time identifies only positive classes with 86% confidence. That's what I think ;)

```
def prediction_pipeline(path =
'../input/tide-match-data/data_interview_test.csv', pca_path =
'pca.pkl', model_path = 'xgb_classifier1.pkl', output_dir = './'):

    test_data_path = data_creation(path, mode = 'test')
    test_data_path = feature_creation(test_data_path , mode = 'test')
    df6 = pd.read_csv(test_data_path)
    feats = list(df6.columns)
    pca = pkl.load(open(pca_path,'rb'))
    X_pca = pca.transform(df6[feats])

    X_pca = pd.DataFrame(X_pca, columns=['pca_'+ str(i) for i in
range(50)], index=df6.index)
    model = joblib.load(model_path)
    preds_proba = model.predict_proba(X_pca)[:,1]
    #print(sum(preds_proba>.3), sum(preds_proba>.2),
sum(preds_proba>.5))

    return preds_proba
```



Receipt - Matching -Pipeline