Rain Prediction

Big Data Project

Contents

1. Team Members

This project is made by *Atudore Darius*, *Bouruc Liviu*, *Frâncu Richard* and *Tender Laura*. All of us follow the Data Science Master's degree (411 group).

2. Introduction

The problem we are trying to solve is predicting if it rains the next day considering more parameters. So, we are trying to solve a binary classification problem.

We use the "Australia Weather Data" dataset. For each entry we have the following list of 22 attributes:

1. Location (category)
2. MinTemp (range)
3. MaxTemp (range)
4. Rainfall (range)
5. Evaporation (range)
6. Sunshine (range)
7. WindGustDir (category)
8. WindGustSpeed (range)
9. WindDir9am (category)
10. WindDir3pm (category)
11. WindSpeed9am (range)
12. WindSpeed3pm (range)

13. Humidity9am (range)
14. Humidity3pm (range)
15. Pressure9am (range)
16. Pressure3pm (range)
17. Cloud9am (range)
18. Cloud3pm (range)
19. Temp9am (range)
20. Temp3pm (range)
21. RainToday (binary)
22. RainTomorrow (binary)

The dataset has about 100k labeled entries. 77.53% of the data is labeled as 0 while 22.46% is labeled as 1. So we work with an unbalanced dataset.

As the dataset is quite big the training time increases when using dimension reduction algorithms. So we decided to only work with 15% of the data (~12k entries). We split it into 80% training and 20% testing data. The ratio of data labeled as zero and data labeled as one stays the same. The reduced dataset has **1.35** MB.

In this project, in order to solve this binary classification problem, we used 4 different models: Random Forest, SVM, Logistic Regression and XGBoost and the following dimension reduction algorithms: PCA, Isomap, Locally Linear Embedding, Kernel PCA.

3. Data preprocessing

By looking at the dataset we notice we have empty cells so we replace empty cells with 0 using fillna method.

```
data = pd.read_csv("/content/Weather Training Data.csv", delimiter = ',')
data.head()
```

	row ID	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity9am	Humidity3pm
0	Row0	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	71.0	22.0
1	Row1	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	44.0	25.0
2	Row2	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	82.0	33.0
3	Row3	Albury	14.6	29.7	0.2	NaN	NaN	WNW	56.0	W	...	55.0	23.0
4	Row4	Albury	7.7	26.7	0.0	NaN	NaN	W	35.0	SSE	...	48.0	19.0

5 rows × 23 columns

We can also notice that `'Location'`, `'WindGustDir'`, `'WindDir9am'`, `'WindDir3pm'`, `'RainToday'` are categories and have values of type string so we use LabelEncoder from sklearn.preprocessing to turn them into numerical values.

```
CAT_COLUMNS = ['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm']
le = LabelEncoder()
data[CAT_COLUMNS] = data[CAT_COLUMNS].apply(le.fit_transform)
```

```
data.fillna(0, inplace=True)
data.RainToday = data.RainToday.map({'Yes': 1, 'No': 0, 0: 0})
```

Now we create the training and testing data and labels using train_test_split twice, once to reduce the dataset entries and second to actually split the data.

```
y = data['RainTomorrow']
X = data.drop(columns=['row ID', 'RainTomorrow'])
alpha = 0.15 # percentage of dataset used
beta = 0.8 # percentage of training data

all_data, _, all_labels, _ = train_test_split(X, y, train_size = alpha, random_state = 28)
X_train, X_test, y_train, y_test = train_test_split(all_data, all_labels, train_size = beta, random_state = 42)
```

4. Models and fine-tuning

4.1 Logistic Regression

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. It is used to predict a binary outcome (1/0, Yes/No, True/False).

We used Logistic Regression from Sklearn library. Sklearn makes available for us a large set of parameters that can be used to improve our training, we restrict our work only to two parameters:

- C - the 'C' parameter is the inverse of regularization strength. Smaller values of C specify stronger regularization. Regularization is used to prevent overfitting, by adding a penalty term to the loss function.
- penalty: - the 'penalty' parameter specifies the type of regularization used in the model. The options are 'l1', 'l2', 'elasticnet' and 'none'. 'l1' penalty is L1 regularization, 'l2' penalty is L2 regularization and 'elasticnet' is a combination of both.

For this project we also used:

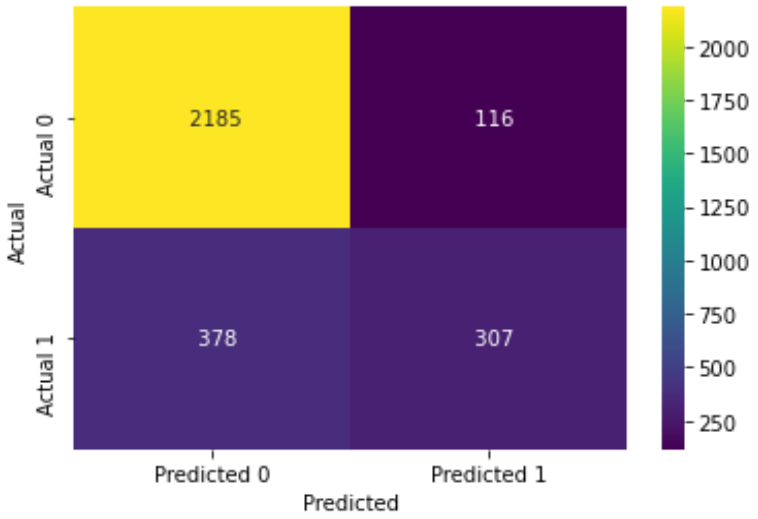
- GridSeachCV (sklearn) - function in the sklearn that is used to tune hyperparameters for a model by training the model multiple times with different combinations of parameter values. It takes a model, a set of parameter values to try, and a scoring method, and returns the best set of parameters as determined by the scoring method.
- StandardScaler (sklearn) - preprocessing method in sklearn that standardizes a feature by subtracting the mean and scaling to unit variance. It is used to transform the feature so that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one

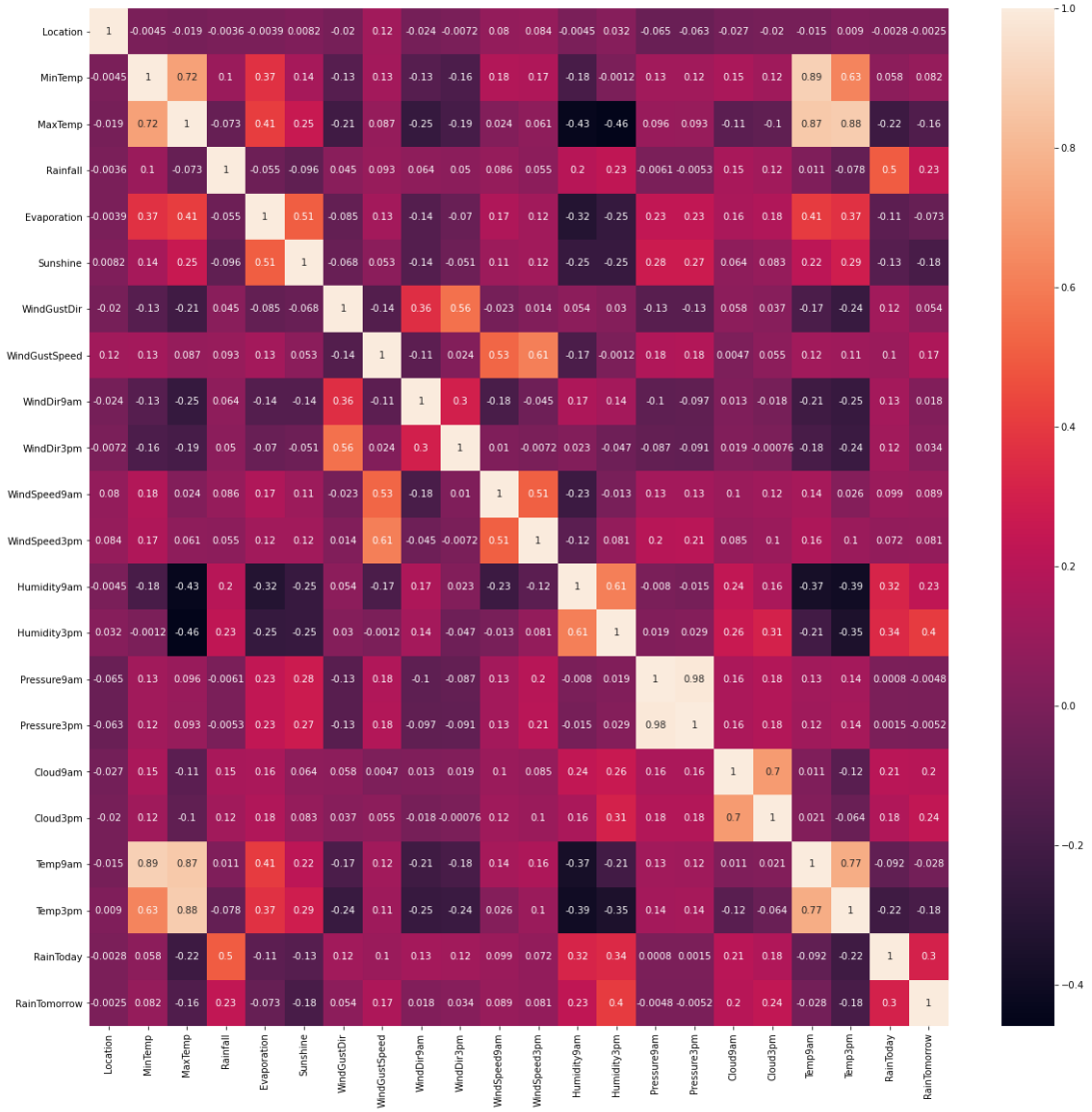
4.1.1 Baseline Model

For the first experiment we used almost all of the features. In further steps we will use this correlation matrix along with other techniques to select a subset of features. Also, we need to specify that for a correct interpretation we have to consider the absolute value when selecting the highest correlation scores.

Given the features below, we simply train our first model and we obtain the baseline

- f1_score: 0.55


2185
116
378
307
Actual 0
Actual 1
Actual
Predicted 0
Predicted 1
Predicted
2000
1750
1500
1250
1000
750
500
250


Location
MinTemp
MaxTemp
Rainfall
Evaporation
Sunshine
WindGustDir
WindGustSpeed
WindDir9am
WindDir3pm
WindSpeed9am
WindSpeed3pm
Humidity9am
Humidity3pm
Pressure9am
Pressure3pm
Cloud9am
Cloud3pm
Temp9am
Temp3pm
RainToday
RainTomorrow
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

4.1.2 Feature Selection and Data Resampling

Feature selection is the process of selecting a subset of features from the original set. Usually this selection is made as a result of different techniques of data analysis, or by training analysis, we can consider the set of features as a hyperparameter.
For our case we selected the set of features based on the correlation with the target feature:

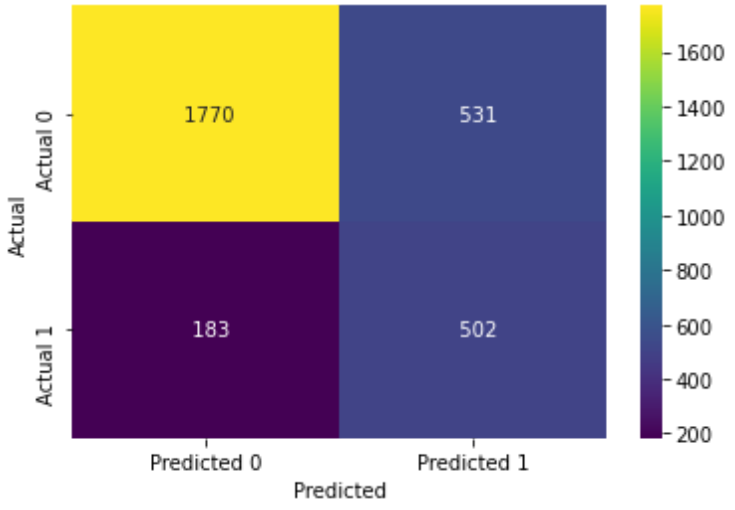
```
FEATURE_COLUMN = ['MinTemp', 'MaxTemp', 'Rainfall','Sunshine',
                  'WindGustSpeed', 'Humidity9am', 'Humidity3pm',
                  'Pressure9am', 'Pressure3pm', 'Cloud9am',
                  'Cloud3pm', 'Temp9am', 'Temp3pm', 'RainToday']
```

Data resampling is the process of manipulating the training data in a machine learning model to improve the model's performance. There are two main types of data resampling: oversampling and undersampling.

For this project we choose oversampling which involves increasing the number of instances in the minority class in the training data, while undersampling involves reducing the number of instances in the majority class.

To solve our problem we use a very useful class from imblearn library, called SMOTE (Synthetic Minority Over-sampling Technique). The technique works by synthesizing new minority instances between existing minority instances, rather than simply duplicating existing instances. SMOTE selects two or more similar minority instances and interpolates new synthetic examples along the line segments joining the selected examples.

- f1_score: 0.58

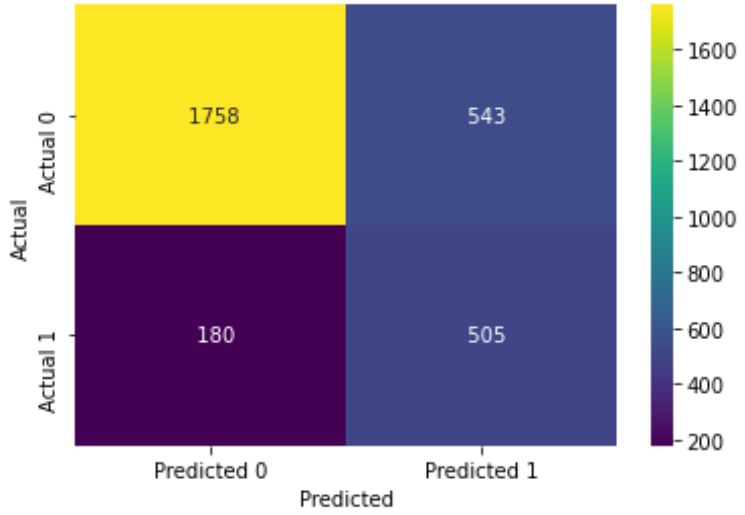


4.1.3 Dimension Reduction

4.1.3.1 PCA

Experiment PCA with multiple number of components and we obtained following f1 scores:

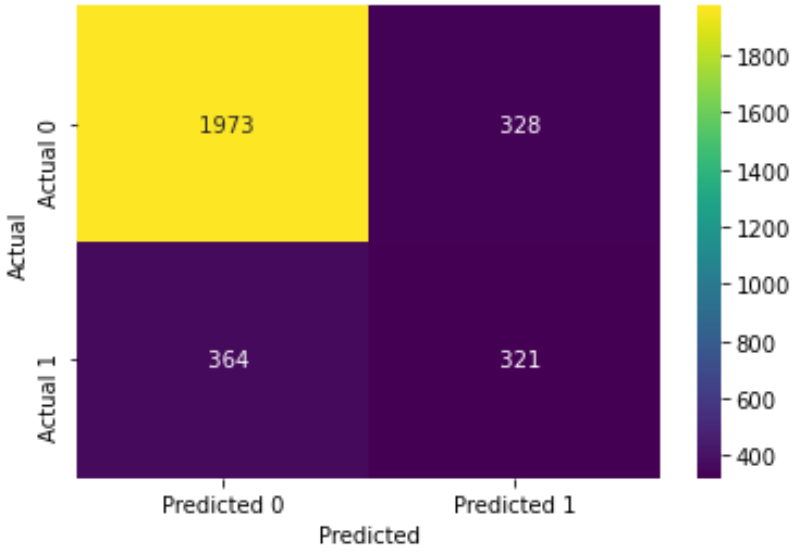
- n_components = 2 : 0.52
- n_components = 5: 0.54
- n_components = 10: 0.58



4.1.3.2 ISOMAP

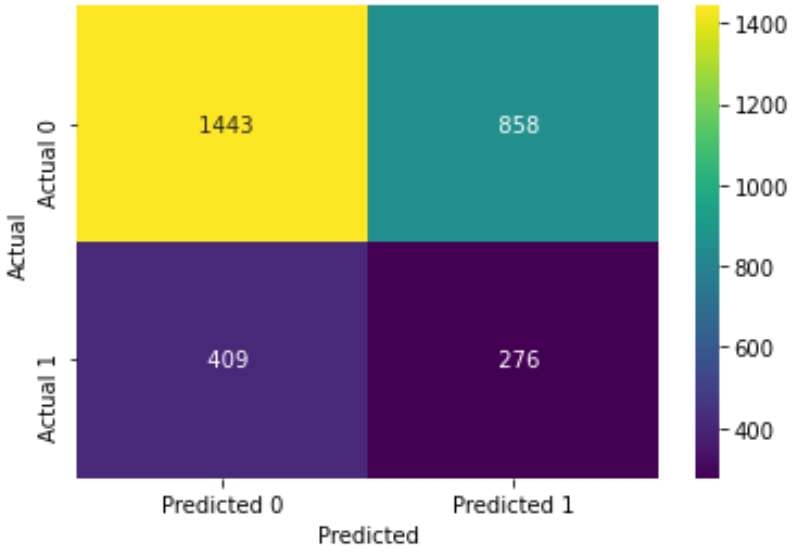
Experiment with Isomap was performed only using default parameters and we obtained:

- f1_score: 0.48

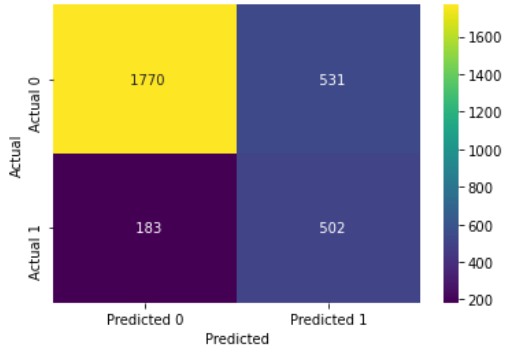
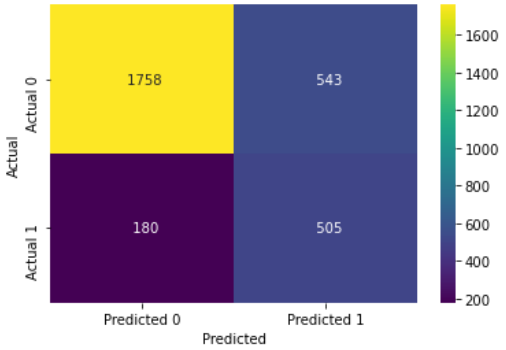
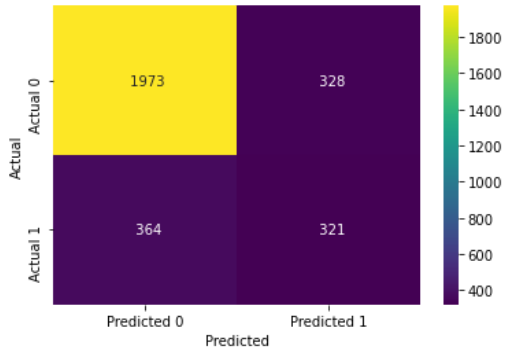
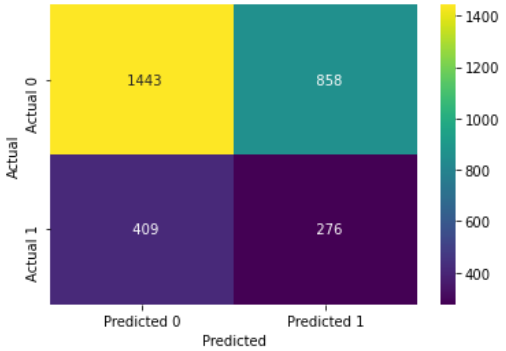


4.1.3.3 Locally Linear Embedding

Experiment with locally linear embedding was performed only using default parameters and we obtained: f1_score: 0.28



CM COMPARISON TABLE

Baseline without dim reduction  f1_score: 0.58	PCA  f1_score: 0.58
ISOMAP  f1_score: 0.48	Locally Linear Embedding  f1_score: 0.28

4.2 Random Forest

Baseline

In this project, we used Random Forest as a classification model. It consists of many decision trees whose low correlated predictions as a whole are more accurate than only one tree's. A decision tree is an algorithm in which each node splits the data by a feature and each leaf node represents a class label.

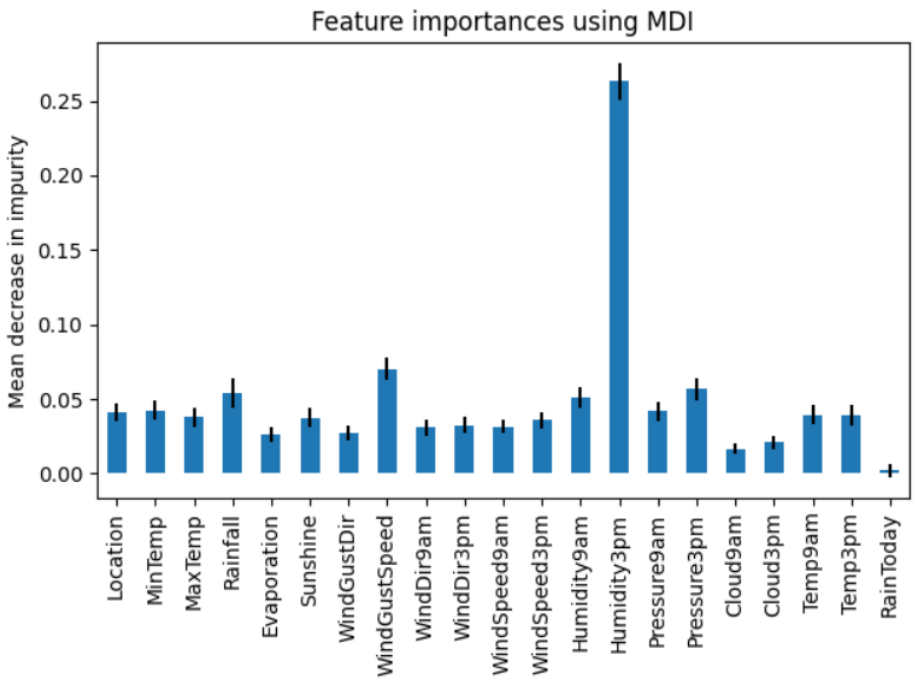
In this case, our baseline model is a Random Forest Classifier from the Sklearn library using two parameters: n_estimators and max_features. n_estimators represents the number of trees in the forest while max_features is the number of features to consider when looking for the best split. The parameters we used are 250 and 21.

```
Bag = RandomForestClassifier(n_estimators = 250, max_features = 21)
Bag = Bag.fit(X_train, y_train)

importances = Bag.feature_importances_
forest_importances = pd.Series(importances, index = X.columns)

# We'll use standard deviation of the values obtained above
std = np.std([tree.feature_importances_ for tree in Bag.estimators_], axis = 0)

# Plotting the graph
fig, ax = plt.subplots()
forest_importances.plot.bar(yerr = std, ax = ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```



Let's have a look at a list of feature importances sorted in descending order.

Humidity3pm	0.263107
WindGustSpeed	0.070200
Pressure3pm	0.056663
Rainfall	0.053965
Humidity9am	0.051337
MinTemp	0.042556
Pressure9am	0.041870
Location	0.041507
Temp9am	0.039512
Temp3pm	0.039224
MaxTemp	0.037993
Sunshine	0.037617
WindSpeed3pm	0.035919
WindDir3pm	0.032598
WindSpeed9am	0.031554
WindDir9am	0.031051
WindGustDir	0.027125
Evaporation	0.026345
Cloud3pm	0.020806
Cloud9am	0.016797
RainToday	0.002254

In this case we notice Humidity at 3 pm is by far the most important feature in predicting if it will rain tomorrow. Next the most important features (importance over >0.05) are wind speed, the pressure at 3pm, rainfall and the humidity at 9 am. Otherwise we don't notice other features standing out. We would have expected RainToday to influence the output more.

These are the metrics obtained by the baseline and the confusion matrix.

	Model	accuracy	precision	recall	f1
0	Random Forest baseline	0.845613	0.740343	0.50365	0.599479

PCA

I analyzed the performance of PCA for our dataset by fine tuning n_component and svd_solver and comparing the model performance by F1 score.

```
pca = PCA(n_components = n_components, random_state = 42)
pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
Bag = RandomForestClassifier(n_estimators = 250, max_features = n_components)
Bag = Bag.fit(X_train_pca, y_train)
y_test_pca = Bag.predict(X_test_pca)
m_f1_score = f1_score(y_test, y_test_pca)
print(f'PCA with n_components {n_components} has f1 score {m_f1_score}')
```

Results:

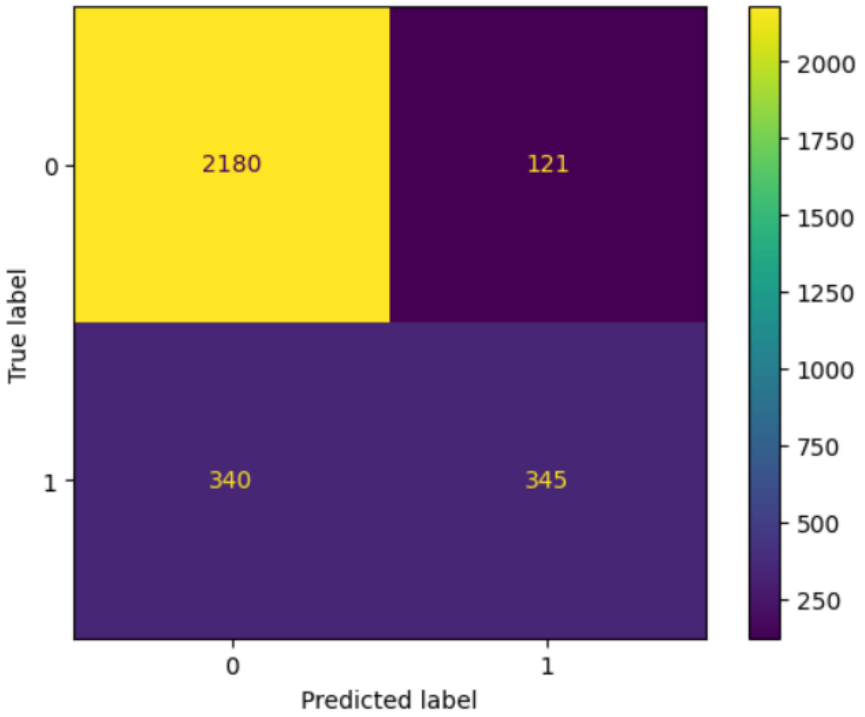
n_components	F1 Score
3	0.4823
5	0.5269
10	0.5775
15	**0.5806**

svd_solver	F1 Score
full	**0.5828**
arpack	0.5764
randomized	0.5765

So the best PCA for our model is:

```
pca = PCA(n_components = 15, svd_solver = "full", random_state = 42)
```

Results:



	Model	accuracy	precision	recall	f1
0	PCA	0.83992	0.724512	0.487591	0.582897

Isomap

I analyzed the performance of Isomap for our dataset by fine tuning n_component and comparing the model performance by F1 score.

```
n_components_tries = [3, 5, 10, 15]
for n_components in n_components_tries:
  isomap = Isomap(n_components = n_components)
  isomap.fit(X_train)
  X_train_isomap = isomap.transform(X_train)
  X_test_isomap = isomap.transform(X_test)
  Bag = RandomForestClassifier(n_estimators = 250, max_features = n_components)
  Bag = Bag.fit(X_train_isomap, y_train)
  y_test_isomap = Bag.predict(X_test_isomap)
  m_f1_score = f1_score(y_test, y_test_isomap)
  print(f'Isomap with n_components {n_components} has f1 score {m_f1_score}')
```

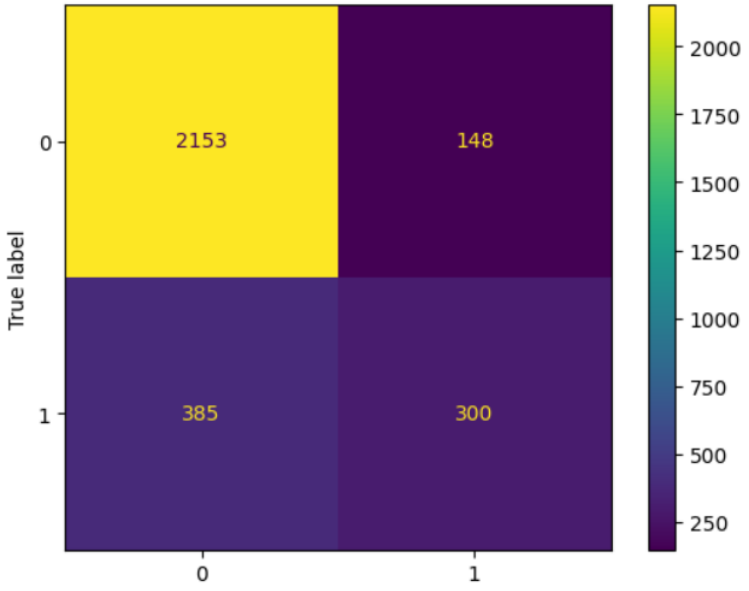
n_components	F1 Score
3	0.4688
5	0.5279
10	0.5319
15	**0.5356**

So the best Isomap for our model is:

```
isomap = Isomap(n_components = 15)
```

Results:

	Model	accuracy	precision	recall	f1
0	Isomap	0.8215	0.669643	0.437956	0.529568



<u>Locally Linear Embedding</u>

I analyzed the performance of PCA for our dataset by fine tuning n_componentand comparing the model performance by F1 score.

```
n_components_tries = [3, 5, 10, 15]

for n_components in n_components_tries:
    lle = LocallyLinearEmbedding(n_components = n_components)
    lle.fit(X_train)

    X_train_lle = lle.transform(X_train)
    X_test_lle = lle.transform(X_test)

    Bag = RandomForestClassifier(n_estimators = 100, max_features = n_components)
    Bag = Bag.fit(X_train_lle, y_train)

    y_test_lle = Bag.predict(X_test_lle)
    m_f1_score = f1_score(y_test, y_test_lle)

    print(f'Locally Linear Embedding with n_components {n_components} has f1 score {m_f1_score}')
```

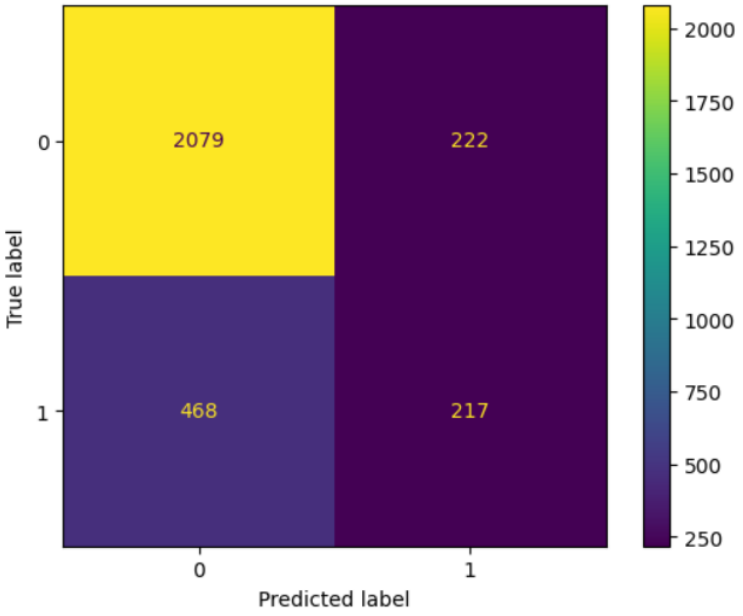
n_components	F1 Score
3	0.0028
5	0.0028
10	0.2066
15	**0.3766**

So the best Locally Linear Embedding for our model is:

```
lle = LocallyLinearEmbedding(n_components = 15)
```

Results:

Model	accuracy	precision	recall	f1
Locally Linear Embedding	0.768922	0.494305	0.316788	0.386121



Kernel PCA

I analyzed the performance of Kernel PCA for our dataset by fine tuning n_component and kernel and comparing the model performance by F1 score.

```
n_components_tries = [3, 5, 10, 15]
kernels = ['linear', 'poly', 'rbf', 'sigmoid', 'cosine']

for n_components in n_components_tries:
    for kernel in kernels:
      kernel_PCA = KernelPCA(n_components = n_components, kernel = kernel)
      kernel_PCA.fit(X_train)

      X_train_pca = kernel_PCA.transform(X_train)
      X_test_pca = kernel_PCA.transform(X_test)

      Bag = RandomForestClassifier(n_estimators = 100, max_features = n_components)
      Bag = Bag.fit(X_train_pca, y_train)

      y_test_pca = Bag.predict(X_test_pca)
      m_f1_score = f1_score(y_test, y_test_pca)

      print(f'PCA with kernel {kernel} and n_components {n_components} has f1 score {m_f1_score}')
```

Results:

	3	5	10	15
linear	0.5022	0.5342	0.5867	**0.5895**
poly	0.5144	0.5763	0.5788	0.5859
rbf	0.0	0.0	0.3731	0.0

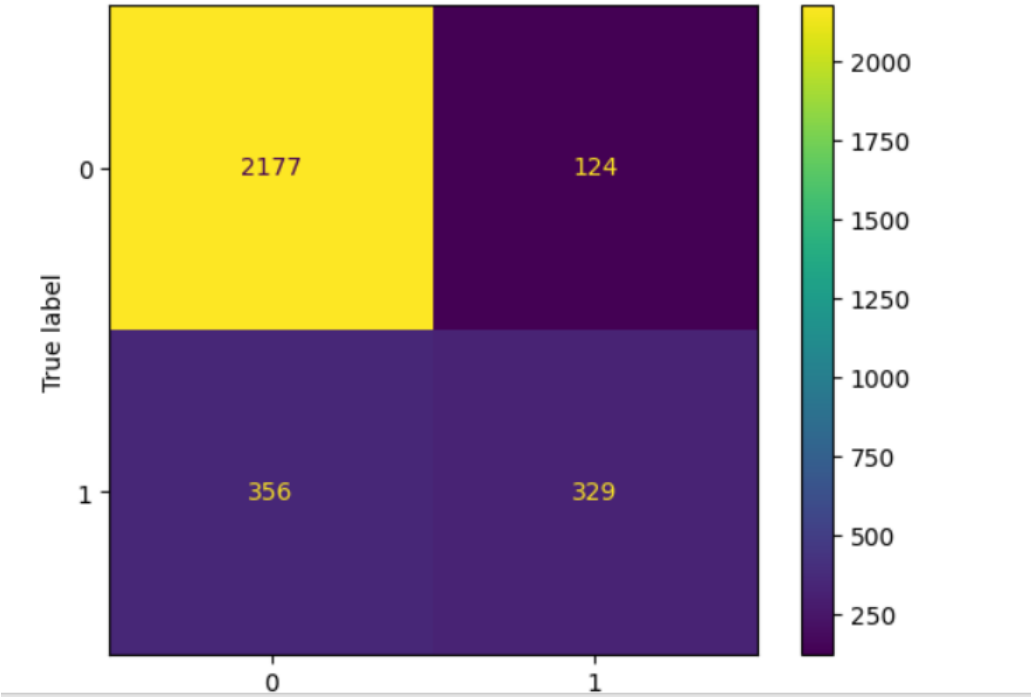
sigmoid	0.0029	0.0029	0.0029	0.0058
cosine	0.4986	0.5427	0.5686	0.5793

So the best PCA for our model is:

```
kernel_PCA = KernelPCA(n_components = 15, kernel = 'linear')
```

Results:

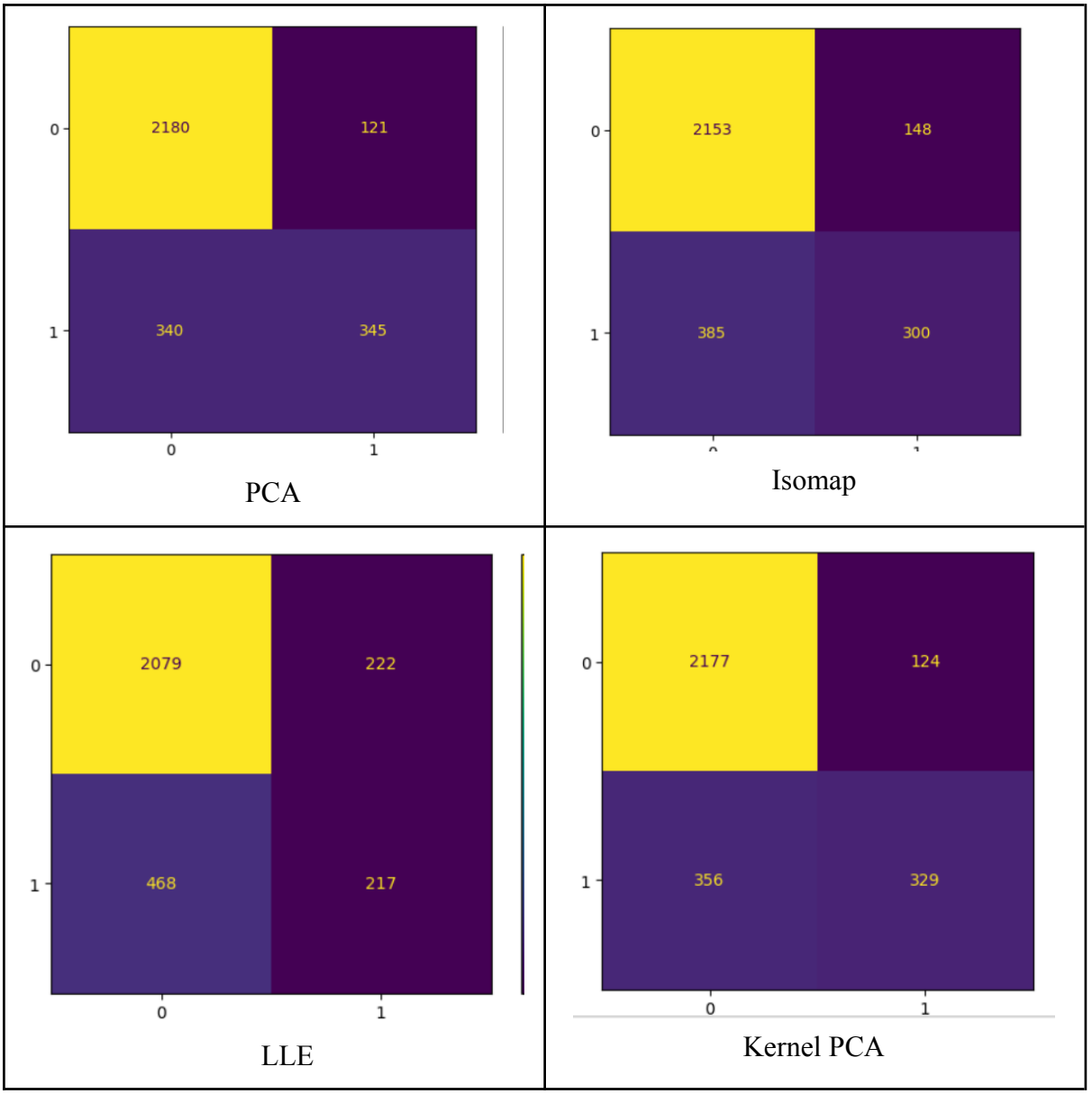
Model	accuracy	precision	recall	f1
Kernel PCA	0.83925	0.726269	0.480292	0.578207



Random Forest Approach Summary

	Accuracy	Precision	Recall	F1
Baseline (21 features)	**0.8456**	**0.7403**	**0.5036**	**0.5994**
PCA (15 features)	0.8399	0.7245	0.4875	0.5828
Isomap (15 features)	0.8215	0.6696	0.4379	0.5295

Locally Linear Embedding (15 features)	0.7689	0.4943	0.3167	0.3861
Kernel PCA (15 features)	0.83925	0.7262	0.4802	0.5782



4.3 SVM

Another machine learning classification model we used is SVM (Support Vector Machines). This is a binary classifier which tries to build a straight delimitation with two margins between two classes as such it tries to maximize the number of data points correctly classified. For the actual implementation, I used the SVC classifier from the SVM module in sklearn.

For the training itself (baseline model), I used a tool named GridSearchCV which lets me make a dictionary with all the hyperparameters I want to test and it runs the model with every combination of them. It also has built-in cross-validation. Hyperparameters tested:

• Regularization parameter *C*: tells the model how much it can wrongly classify data; a large C reduces the learned margins (can lead to overfitting) / a small C is more permissive (can lead to underfitting)

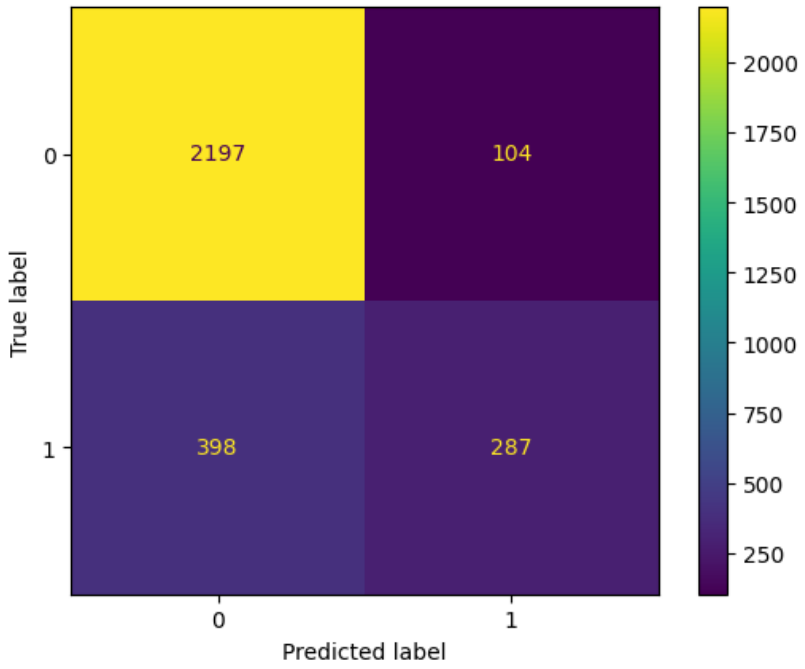
• *kernel* parameter which represents the type of kernel the model uses, his role being to project the data to linearly separable feature spaces

Because the dataset is very imbalanced, the accuracy isn't the best metric to guide on, so we took into account the F1 Score. Analyzing the best result from above (kernel linear and C = 1), we tested the dimensionality reduction algorithms with those parameters.

Baseline

	C=1	C=1.5	C=3
kernel='rbf'	0.0029	0.0454	0.2577
kernel='linear'	**0.5334**	0.5323	0.5313

Pentru C=1 și kernel='linear', matricea de confuzie este următoarea:



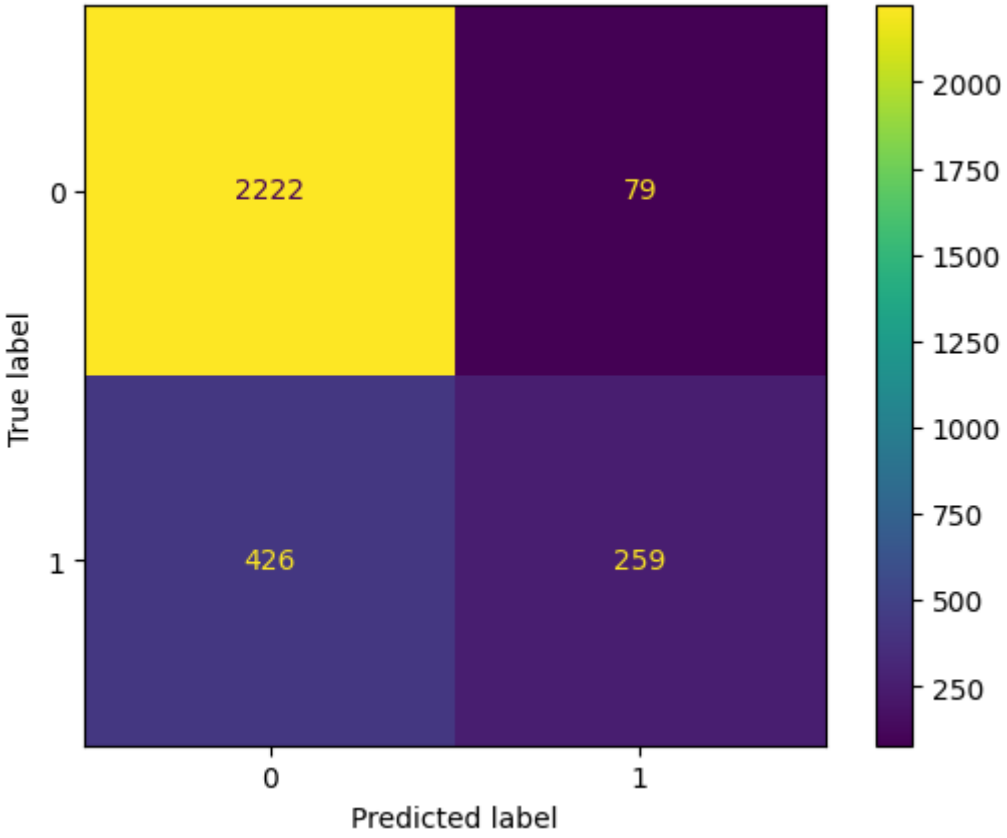
Pentru algoritmii de dimensionality reduction, vom folosi în continuare hyperparametrii de mai sus care au dat rezultatele cele mai bine.

PCA

n_components	F1 Score
3	0.0
5	0.3925
10	0.4526
15	**0.5063**

The best results were obtained with n_components = 15.

	Model	accuracy	precision	recall	f1
0	PCA	0.830877	0.766272	0.378102	0.506354

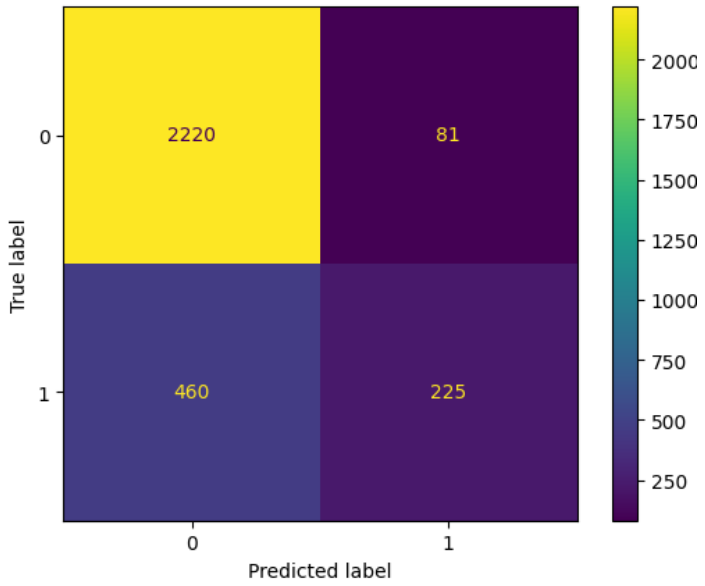


Isomap

n_components	F1 Score
3	0.0
5	0.4008
10	0.4393
15	**0.4540**

The best results were obtained with n_components = 15.

	Model	accuracy	precision	recall	f1
0	Isomap	0.818821	0.735294	0.328467	0.454087

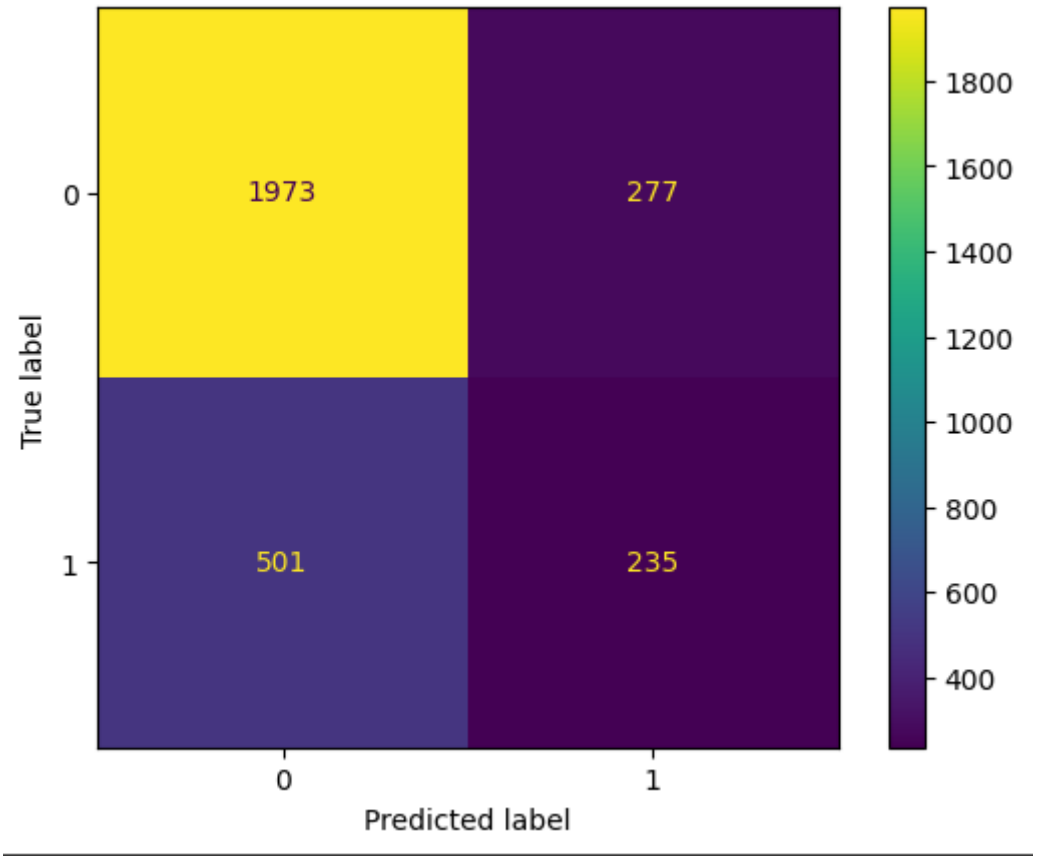


Locally Linear Embedding

n_components	F1 Score
3	0.0029
5	0.0114
10	0.2068
15	**0.3766**

The best results were obtained with n_components = 15.

Model	accuracy	precision	recall	f1
Locally Linear Embedding	0.739451	0.458984	0.319293	0.376603

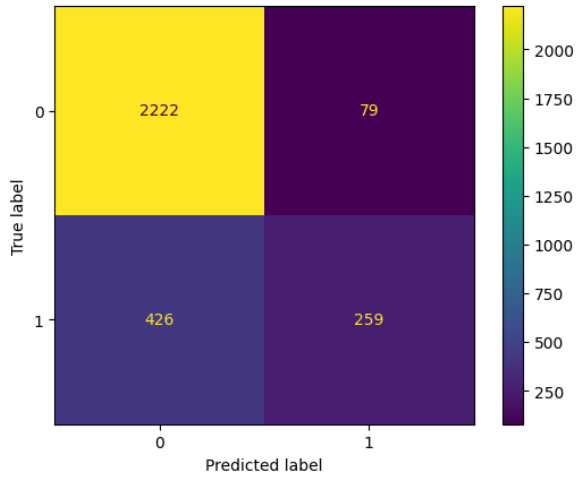


On Kernel PCA, the SVM couldn't converge in time.

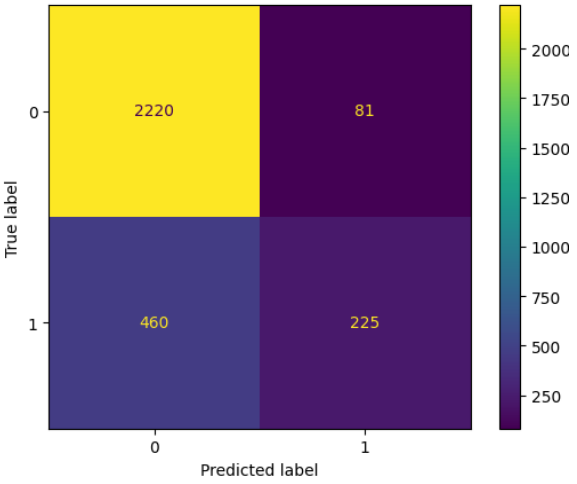
SVM Approach Summary

	Accuracy	Precision	Recall	F1
Baseline (21 features)	**0.8318**	**0.7340**	**0.4189**	**0.5334**
PCA (15 features)	0.8308	0.7662	0.3781	0.5063
Isomap (15 features)	0.8188	0.7352	0.3284	0.4540
Locally Linear Embedding	0.7394	0.4589	0.3139	0.3766

(15 features)				

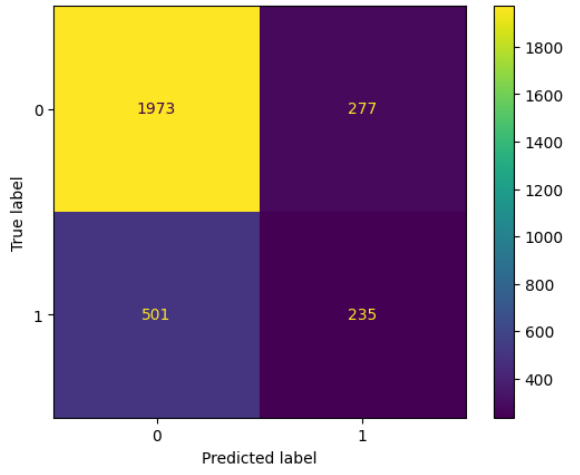


PCA



Isomap

LLE



Kernel PCA

could not converge

4.4 XGBoost

One of the models used in this project is the XGBoost. Also known as Extreme Gradient Boosting algorithm, it is a decision tree based machine learning algorithm which uses a process called boosting to help improve performance. Since its introduction, it's become one of the most effective machine learning algorithms and regularly produces results that outperform most other algorithms, such as logistic regression, the random forest model and regular decision trees.

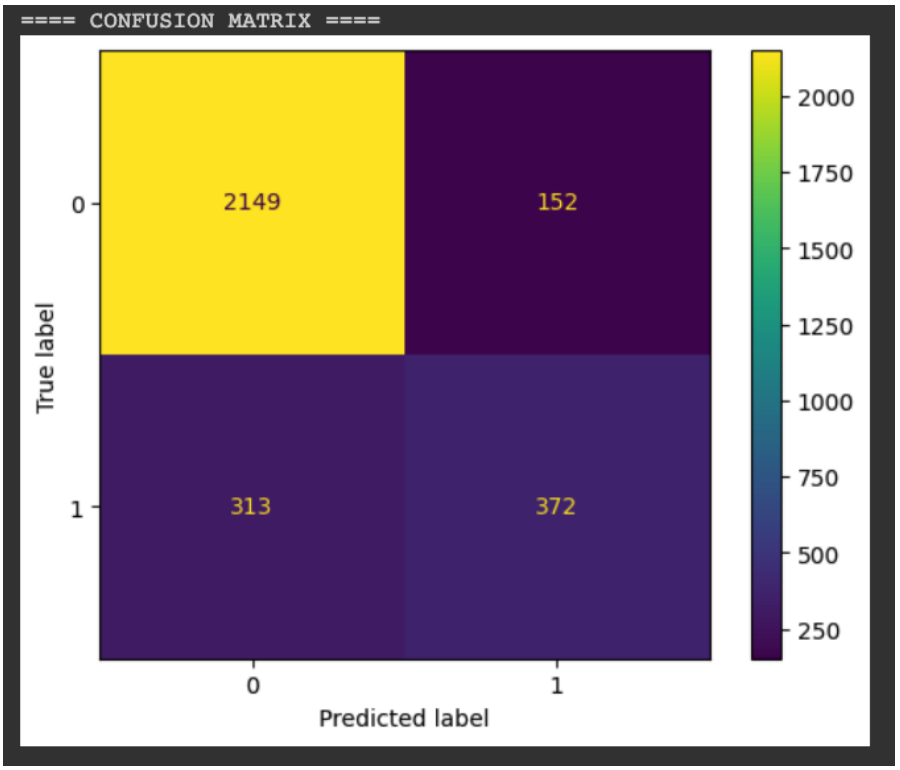
We chose to work with the XGBoost library and its implementation of the XGBClassifier. XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way. The same code runs on a major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

Because of the imbalanced dataset, the metric that we will use to measure how effective the different approaches that we've taken are, we will use the F1 score instead of the accuracy.

For the baseline with the XGBoost algorithm we have the following results:

	Model	accuracy	precision	recall	f1
0	XGBoost	0.844273	0.709924	0.543066	0.615385

With the confusion matrix looking like this:



After we established the baseline, we started fiddling with the four different dimension reduction methods:

PCA

The first method was PCA, where we tried different values for the n_component and svd_solver parameters.
The results are as follows:

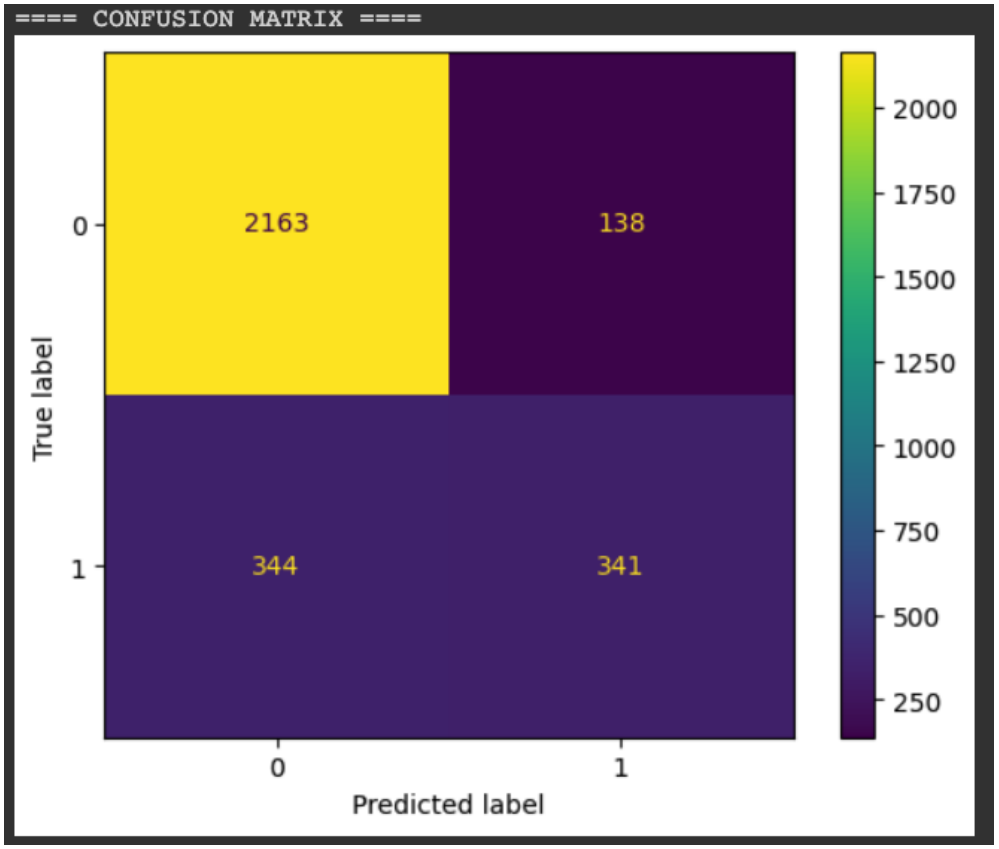
N_COMPONENTS	F1 SCORE
3	0.5193171608265947
5	0.5533807829181494
10	0.5848252344416027
15	0.5743589743589744
16	**0.5859**106529209621
17	0.5789473684210525
18	0.5747899159663865
19	0.5728813559322034

The best result we obtained was with 16 features. Next we tried finding the best svd_solver value for this algorithm:

SVD_SOLVER	F1 SCORE
full	0.5859106529209621
arpack	**0.5859106529209621**
randomized	0.5859106529209621

Because the results were the same, we chose to move forward with the 'arpack' solver (F1 score is **0.585911**).
The complete results for this approach are listed in the image inserted bellow:

	Model	accuracy	precision	recall	f1
0	XGBoost	0.83858	0.7119	0.49781	0.585911



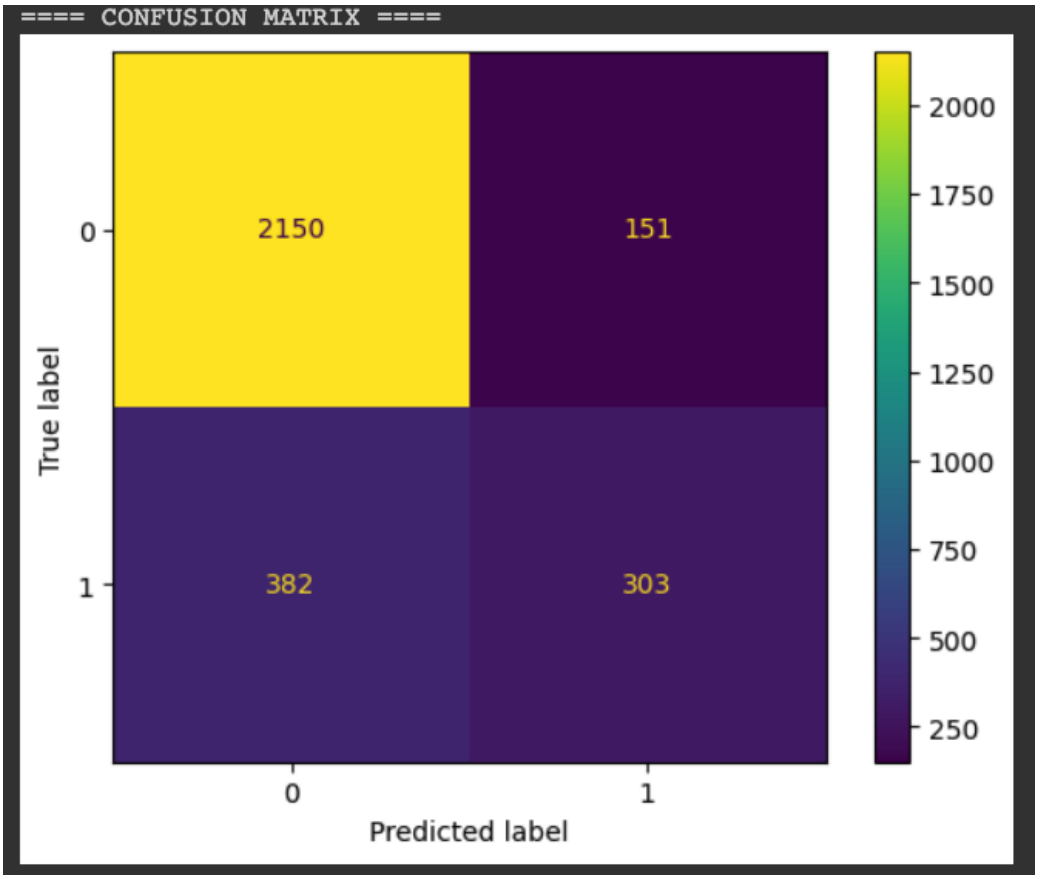
Isomap

The next dimension reduction method that we applied was Isomap. Because it takes quite a long time to compute the results for this one, we first applied a data cut to reduce the dimensions of our data by 85%. Next we tried tuning the n_components parameter to gather the best result.

N_COMPONENTS	F1 SCORE
3	0.44052044609665425
5	0.5145374449339206
10	0.527336860670194
15	0.5235602094240838
18	**0.5320456540825286**

The best result we obtained was with 18 components:

	Model	accuracy	precision	recall	f1
0	XGBoost	0.8215	0.667401	0.442336	0.532046



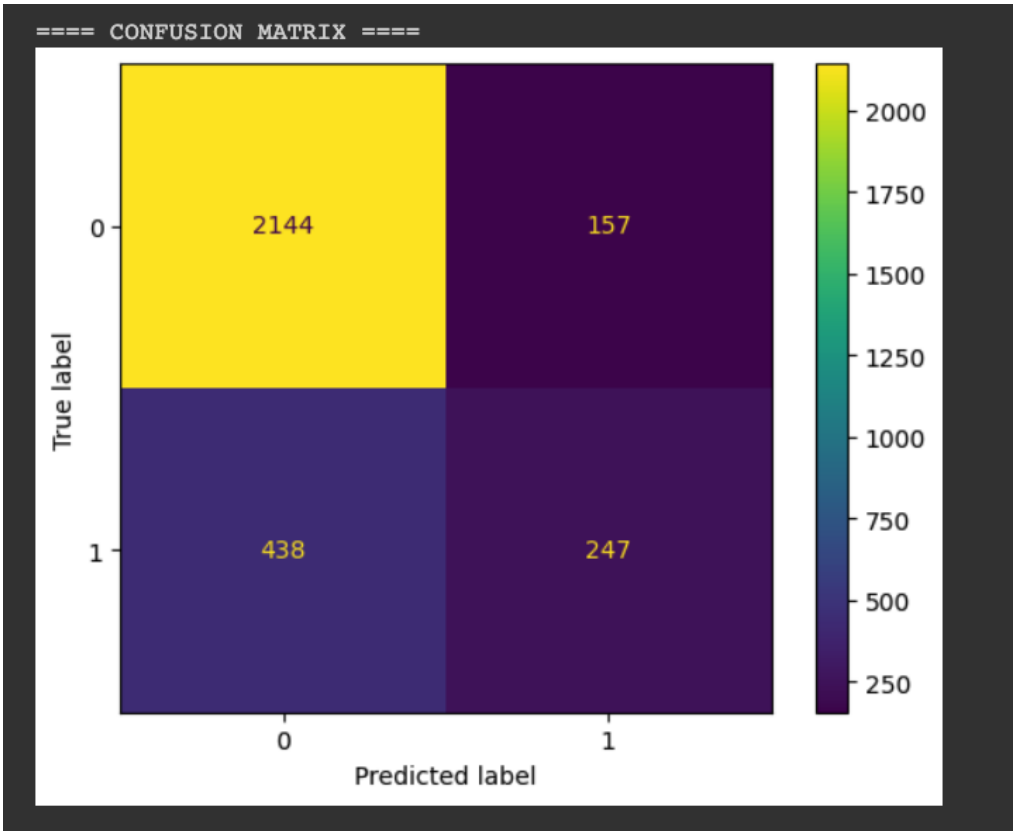
Locally Linear Embedding

The third method was Locally Linear Embedding. Once again the parameter that we tuned was n_components.

N_COMPONENTS	F1 SCORE
5	0.017045454545454548
10	0.19772727272727272
15	0.37101449275362325
18	**0.45362718089990817**

The best results were obtained with 18 components:

	Model	accuracy	precision	recall	f1
0	XGBoost	0.800737	0.611386	0.360584	0.453627



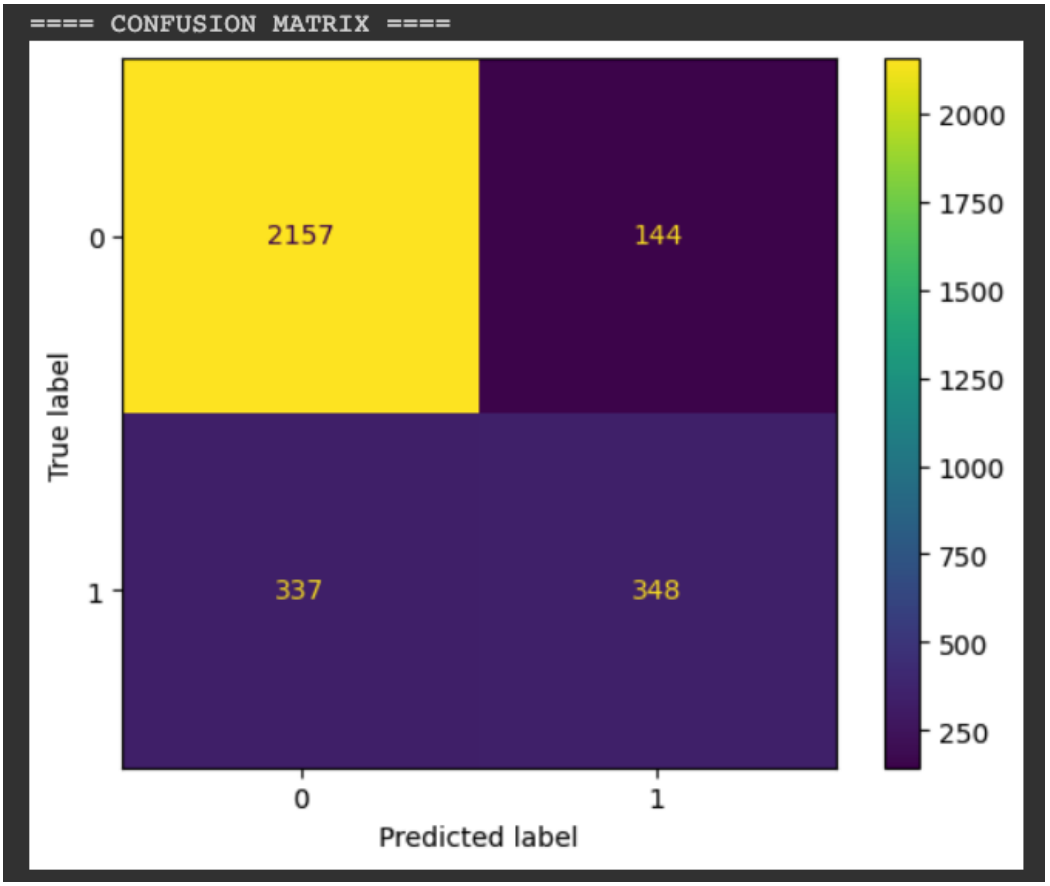
Kernel PCA

The last method used was KernelPca, this time working with the n_components and kernel parameters:

	linear	poly	rbf	sigmoid	cosine
5	0.5533	0.5411	0.0	0.0199	0.5411
10	0.5848	0.5882	0.3721	0.0170	0.5649
15	0.5743	0.5784	0.0028	0.0171	**0.5913**

The best result was with 15 components and cosine kernel:

	Model	accuracy	precision	recall	f1
0	XGBoost	0.838915	0.707317	0.508029	0.591334

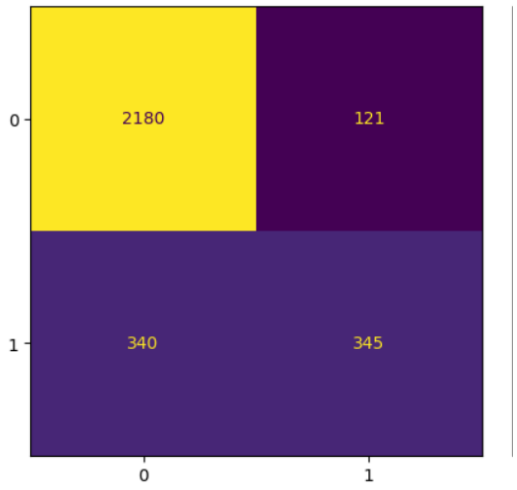


5. Results

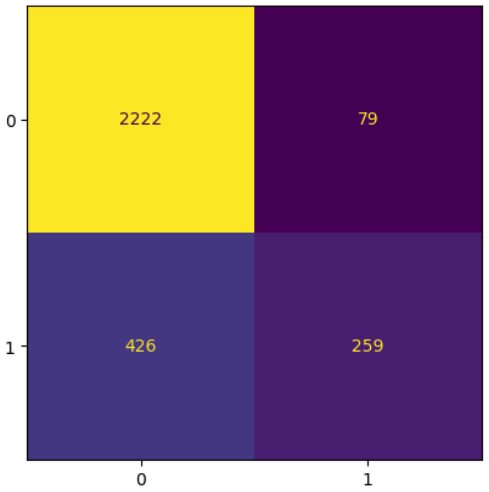
Summary of our best models' results on the test data:

	Random Forest	SVM	Logistic Regression	XGBoost
Baseline (21 features)	0.5994	0.5334	0.58	0.6153
PCA	0.5828 (15 features)	0.5063 (15 features)	0.58 (10 features)	0.5859 (16 features)
Isomap	0.5295 (15 features)	0.4540 (15 features)	0.48 (2 features)	0.5320 (18 features)
LLE	0.3861 (15 features)	0.3734 (15 features)	0.28 (2 features)	0.4536 (18 features)
Kernel PCA	0.5782 (15 features)	_	_	0.5913 (15 features)

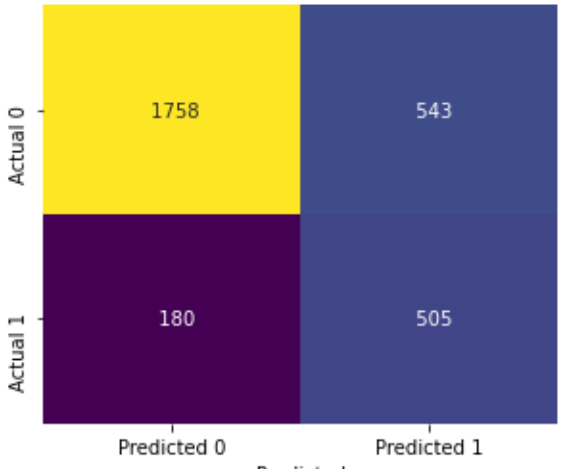
Below we can analyze the confusion matrices using PCA for the results above:



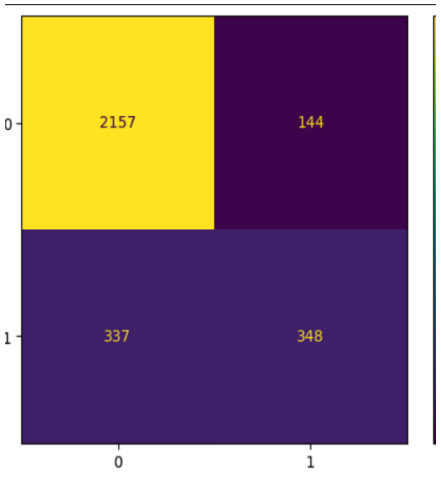
Random Forest using PCA



SVM using PCA



Logistic Regression using PCA



XGBoost using KernelPCA