PYTHON FOR DATA ANALYSIS: POLISH COMPANIES BANKRUPTCY

BY HUGO LABORDE, MATTEO LUSARDI, RAPHAEL MICHON

Introduction

Based on 64 measurements, we predict the health of a company :
Will it stay afloat or will it go bankrupt?

Dataset

The dataset is about bankruptcy prediction of Polish companies.The bankrupt companies were analyzed in the period 2000-2012, while the still operating companies were evaluated from 2007 to 2013.

For each of the 5 years:
64 variables columns with missing values

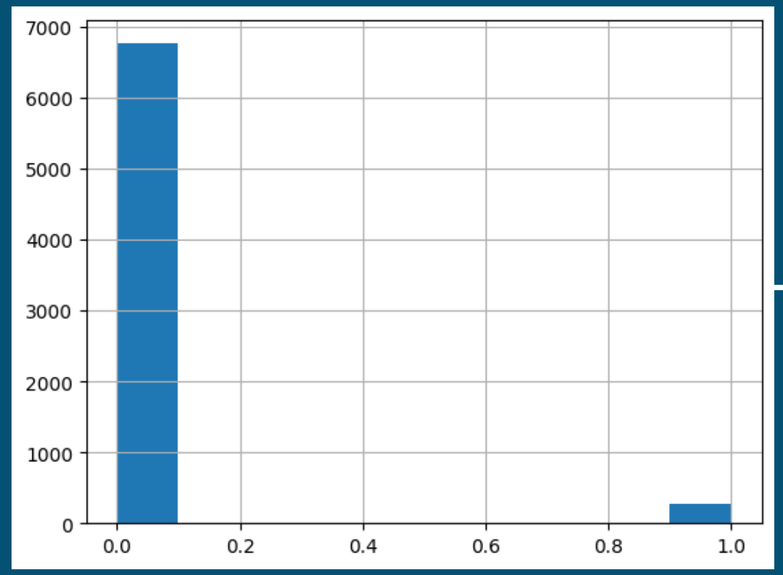
Data cleaning

```
[99] df1.isnull().sum().sort_values()

a predire                                                    0
(gross profit + interest) / sales                            0
net profit / sales                                           0
(inventory * 365) / sales                                    0
gross profit / sales                                         0
                                                          ...
net profit / inventory                                     134
sales / inventory                                          135
profit on operating activities / financial expenses        311
sales (n) / sales (n-1)                                   1622
(current assets - inventories) / long-term liabilities    2740
Length: 65, dtype: int64
```

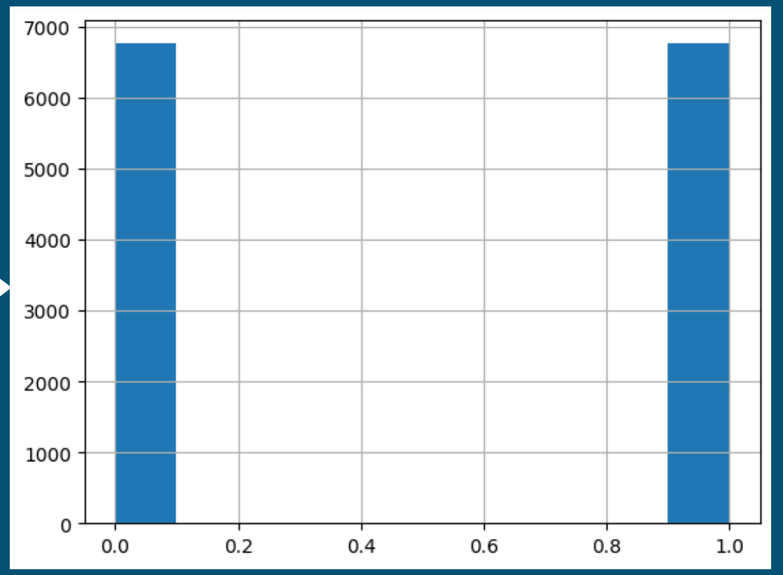
```
knn_imputed_df1.isnull().sum().sort_values()

net profit / total assets                                  0
profit on sales / total assets                             0
total sales / total assets                                 0
(current assets - inventories) / long-term liabilities     0
constant capital / total assets                            0
                                                         ..
working capital / fixed assets                             0
logarithm of total assets                                  0
(total liabilities - cash) / sales                         0
(current liabilities * 365) / cost of products sold        0
a predire                                                  0
Length: 65, dtype: int64
```

```
import fancyimpute

def perform_knn_imputation(df):
    knn_imputed_dataset = fancyimpute.KNN(k=100, verbose=True).fit_transform(df)

    # Convertir le tableau résultant en un dataframe
    knn_imputed_dataframe = pd.DataFrame(data=knn_imputed_dataset, columns=df.columns, index=df.index)

    return knn_imputed_dataframe

knn_imputed_df1 = perform_knn_imputation(df1)
```

SMOTE algo



Classe 0 Classe 1

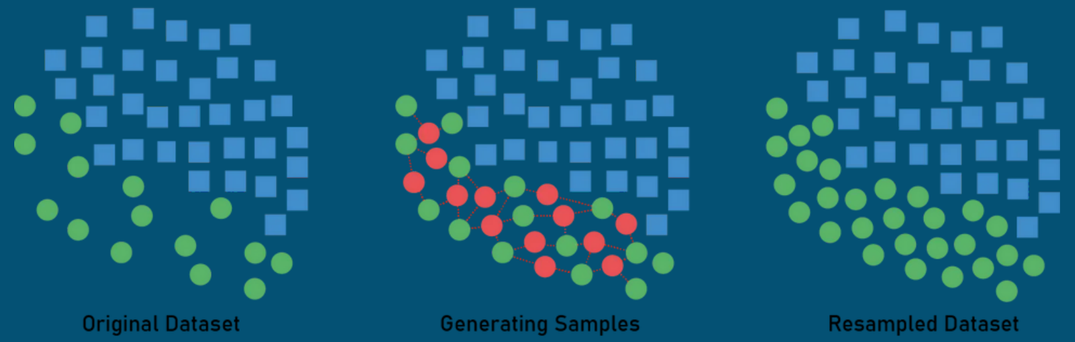


Classe 0 Classe 1

SMOTE algo

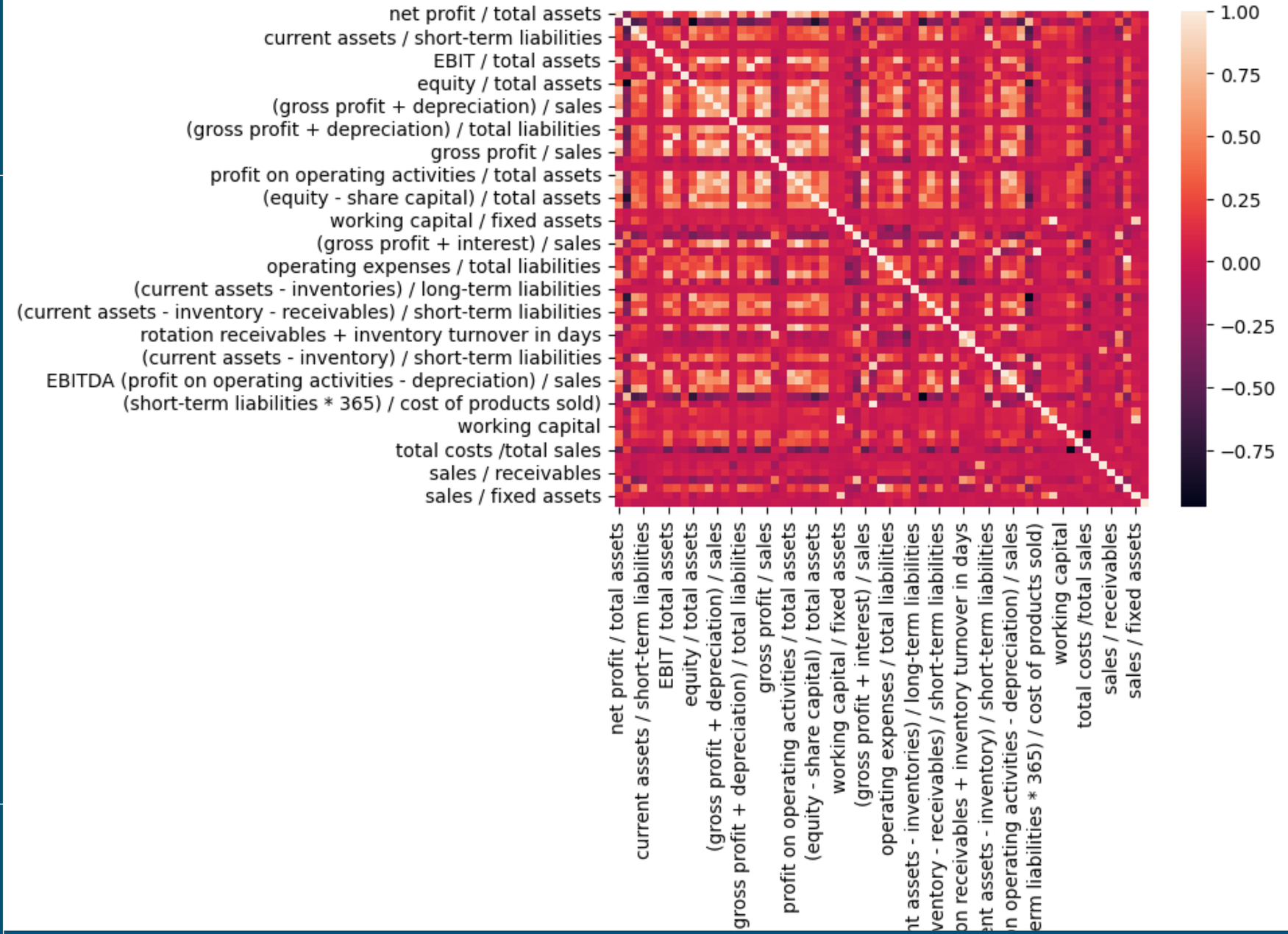
```python
from imblearn.over_sampling import SMOTE

def oversample_with_SMOTE(df, target_column):

    X = df.drop(target_column, axis=1)
    y = df[target_column]                    #Je lance l'algo
    smote = SMOTE()
    X_resampled, y_resampled = smote.fit_resample(X, y)

    resampled_counts = pd.Series(y_resampled).value_counts()
    print(f"Resampled dataset shape {resampled_counts.to_dict()}")  # je verifie la différence de taille
    original_counts = y.value_counts()
    print(f"Original dataset shape {original_counts.to_dict()}")


    resampled_df = pd.DataFrame(X_resampled, columns=X.columns) #Ici je crée la structure du df
    resampled_df[target_column] = y_resampled #Je le remplis

    return resampled_df

oversampled_df1 = oversample_with_SMOTE(knn_imputed_df1, 'a predire')
```

```
Original dataset shape {0.0: 6756, 1.0: 271}
Resampled dataset shape {0.0: 6756, 1.0: 6756}
```




```python
oversampled_df2=oversample_with_SMOTE(perform_knn_imputation(df2),'a predire')
oversampled_df3=oversample_with_SMOTE(perform_knn_imputation(df3),'a predire')
oversampled_df4=oversample_with_SMOTE(perform_knn_imputation(df4),'a predire')
oversampled_df5=oversample_with_SMOTE(perform_knn_imputation(df5),'a predire')
```

Correlation Matrix



ACP

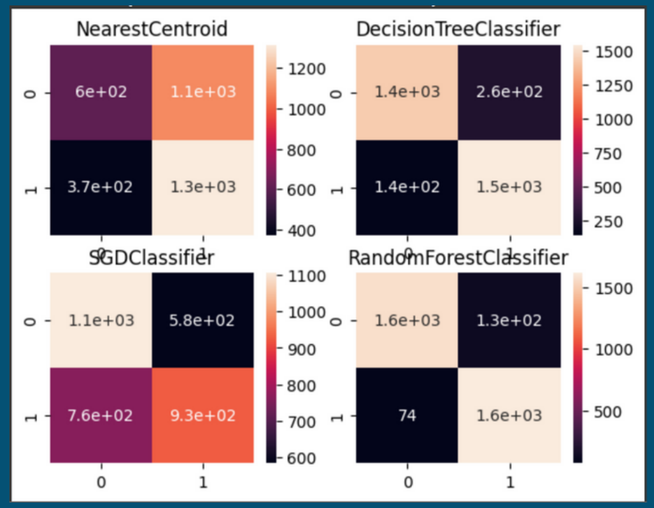
```
from sklearn.decomposition import PCA
# You must normalize the data before applying the fit method
pcadf1=df1.copy()
pcadf1.drop(columns='a predire',inplace=True)
pcadf1.dropna(axis=0,inplace=True)
df1_normalized=(pcadf1 - pcadf1.mean()) / pcadf1.std()
#df1_normalized.dropna(axis=1,inplace=True)
pca = PCA(n_components=pcadf1.shape[1])
pca.fit(df1_normalized)
# Reformat and view results
loadings = pd.DataFrame(pca.components_.T,
columns=['PC%s' % _ for _ in range(len(df1_normalized.columns))],
index=pcadf1.columns)
print(loadings)
plot(pca.explained_variance_ratio_)
ylabel('Explained Variance')
xlabel('Components')
grid()
show()
print(pca.explained_variance_ratio_)
```

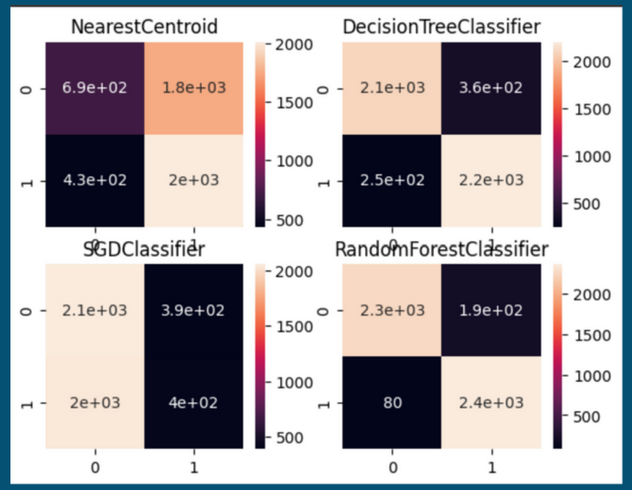
Modeling

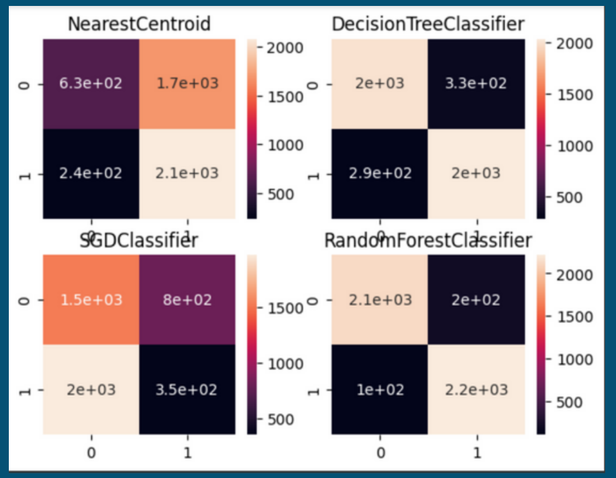
Creation and Training

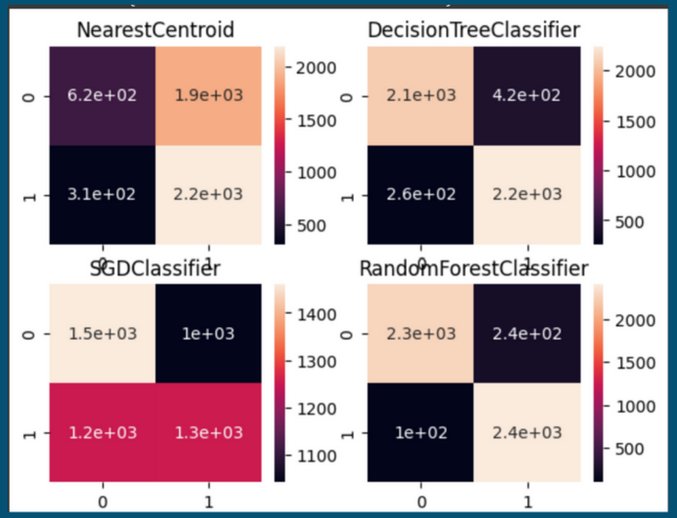
```
pcadf1.drop(columns='a predire',inplace=True)
pcadf1.dropna(axis=0,inplace=True)
df1_normalized=(pcadf1 - pcadf1.mean()) / pcadf1.std()
pca = PCA(n_components=pcadf1.shape[1])
pca.fit(df1_normalized)
comp=pca.explained_variance_ratio_
cum_ex_var_r=[comp[:i].sum() for i in range(len(comp))]
reduc=pca.components_[:pexpdata(cum_ex_var_r,0.99)]
reddf1=oversampled_df1.copy()
reddf1.dropna(axis=0,inplace=True)
apredire=reddf1['a predire']
reddf1.drop(columns='a predire',inplace=True)
reddf1=reddf1.dot(reduc.T)
reddf1['a predire']=apredire
from sklearn.model_selection import train_test_split
X = reddf1.loc[:, reddf1.columns != "a predire"]
y = apredire
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=20)
from sklearn.neighbors import NearestCentroid
clf=NearestCentroid()
clf.fit(X_train, y_train)
pred=clf.predict(X_test)
l1=[]
acc=1-array([abs(list(y_test)[i]-pred[i]) for i in range(len(pred))]).sum()/len(pred)
l1+=[(acc,"NearestCentroid")]
cm1=confusion_matrix(y_test,pred)
param_grid_nearest_centroid = {'metric': ['euclidean', 'manhattan', 'cosine']}
grid_search_nearest_centroid = GridSearchCV(clf, param_grid_nearest_centroid, cv=5)
grid_search_nearest_centroid.fit(X_train, y_train)
print("Meilleurs paramètres pour NearestCentroid:", grid_search_nearest_centroid.best_params_)
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf.fit(X_train, y_train)
pred=clf.predict(X_test)
acc=1-array([abs(list(y_test)[i]-pred[i]) for i in range(len(pred))]).sum()/len(pred)
l1+=[(acc,"DecisionTreeClassifier")]
cm2=confusion_matrix(y_test,pred)
param_grid_decision_tree = {'criterion': ['gini', 'entropy'], 'max_depth': [None, 10, 20, 30]}
grid_search_decision_tree = GridSearchCV(clf, param_grid_decision_tree, cv=5)
grid_search_decision_tree.fit(X_train, y_train)
print("Meilleurs paramètres pour DecisionTreeClassifier:", grid_search_decision_tree.best_params_)
from sklearn.linear_model import SGDClassifier
clf = SGDClassifier(loss="hinge", penalty="l2", max_iter=500)
clf.fit(X_train, y_train)
pred=clf.predict(X_test)
acc=1-array([abs(list(y_test)[i]-pred[i]) for i in range(len(pred))]).sum()/len(pred)
l1+=[(acc,"SGDClassifier")]
cm3=confusion_matrix(y_test,pred)
param_grid_sgd_classifier = {'loss': ['hinge', 'log', 'modified_huber'],
                             'alpha': [0.0001, 0.001, 0.01, 0.1]}
grid_search_sgd_classifier = GridSearchCV(clf, param_grid_sgd_classifier, cv=5)
grid_search_sgd_classifier.fit(X_train, y_train)
print("Meilleurs paramètres pour SGDClassifier:", grid_search_sgd_classifier.best_params_)
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
pred=clf.predict(X_test)
acc=1-array([abs(list(y_test)[i]-pred[i]) for i in range(len(pred))]).sum()/len(pred)
l1+=[(acc,"RandomForestClassifier")]
cm4=confusion_matrix(y_test,pred)
```

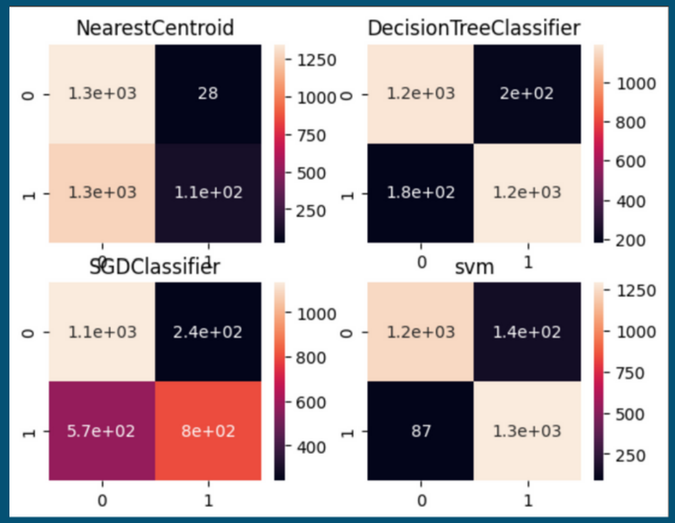
Comparison of models











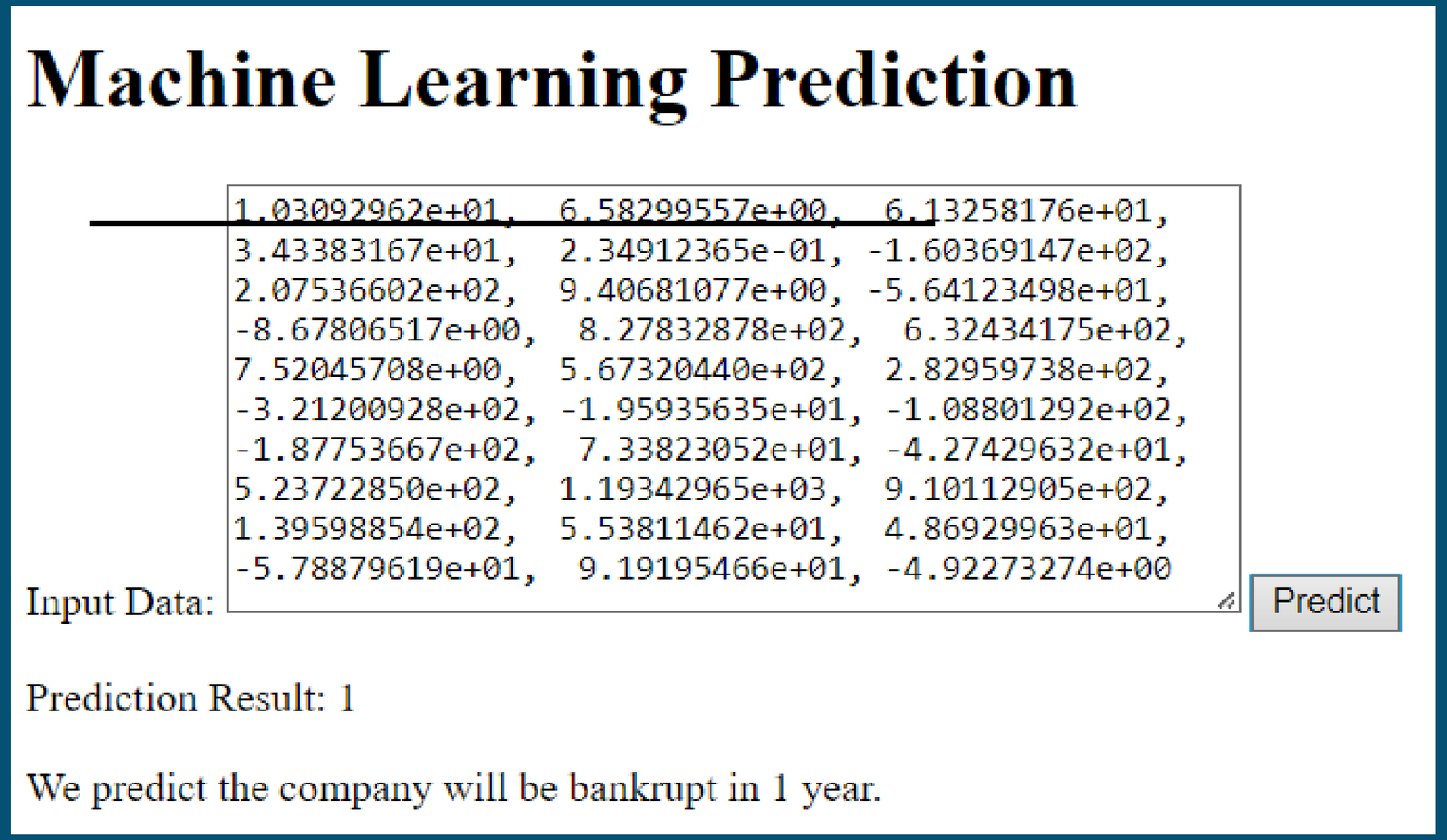
API : Flask

Implementation of a FlaskProject:

```python
class PredictResource(Resource):
    def get(self):
        return {'message': 'Use POST request to submit data for prediction.'}

    def post(self):
        data = request.get_json(force=True)
        input_data = data.get('input_data', None)

        if input_data is None:
            return {'error': 'Input data is missing'}, 400

        try:
            # Parse the string representation of the array
            input_array_str = input_data.strip("[]")  # Remove square brackets
            input_array = [float(value) for value in input_array_str.split(',')]  # Convert to list of floats

            # Ensure that the input_array is a 1D array
            input_array = np.array(input_array).reshape(1, -1)

            # Perform prediction
            prediction = model.predict(input_array)

            return {'prediction': prediction.tolist()}
        except Exception as e:
            return {'error': f'Prediction failed: {str(e)}'}, 500
```

API : Flask

Utilizing the API to predict the bankruptcy of a company:



Thank you for listening

Do you have any questions?