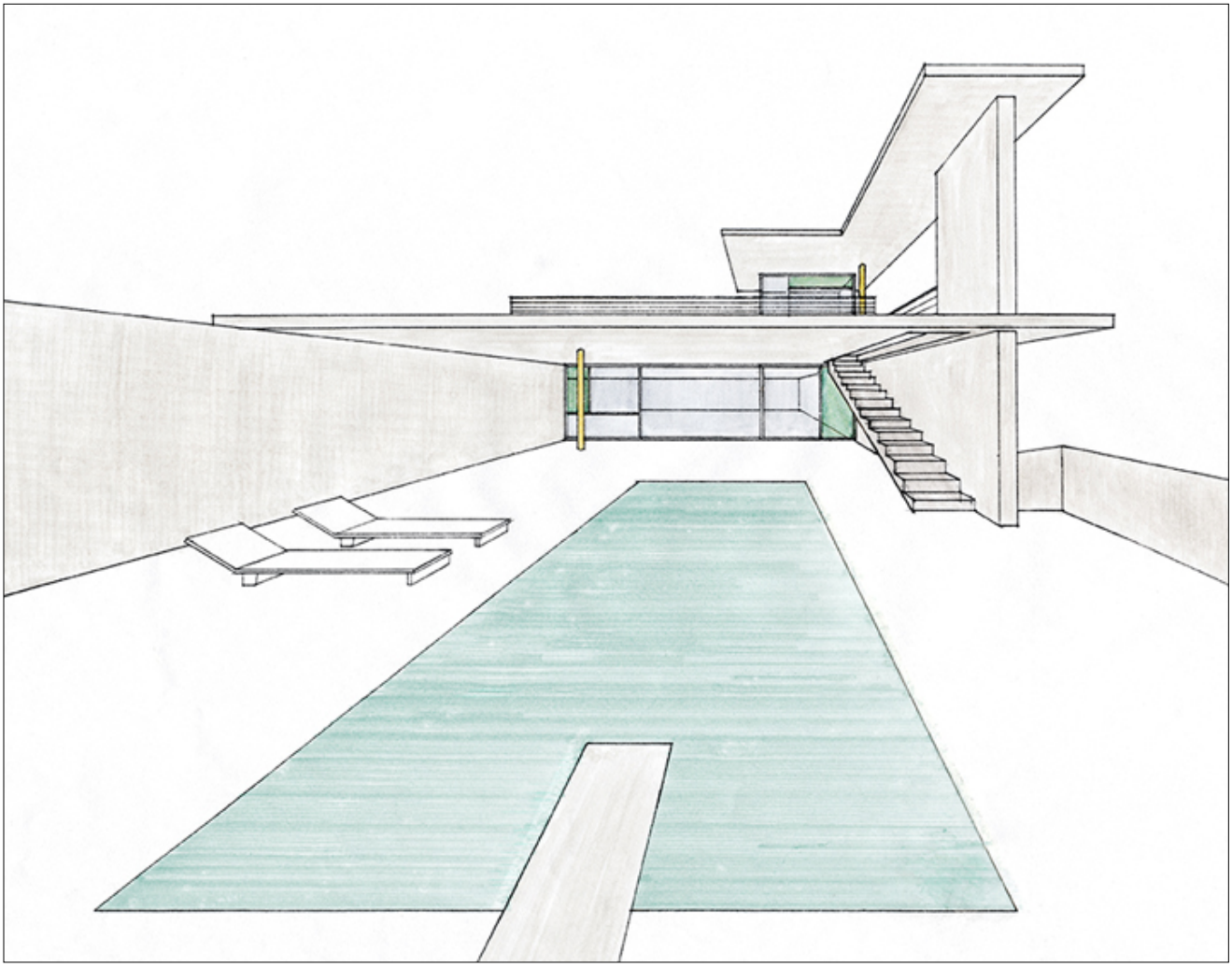
TEAM 4



Python Project II

Risk of Home Equity Line of Credit (HELOC) - A Predictive Model and DSS

Team 4 - Yixuan Gao, Yachao Hu, Wenjie Li, Jinying Xue, Shirley Zheng

CONTENT

IMPORT LIBRARIES & DATA CLEANING

Import Necessary Libraries

First off, we need to import several python libraries including: numpy and pandas for data processing, sklearn for model training, pickle for serializing and streamline for building interface.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import Imputer
from sklearn import svm
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import neighbors
from sklearn.linear_model import LogisticRegression
import pickle
import streamlit as st
```

Read Data

Then, we read the csv file using pd.read_csv.

```
# read the data
data = pd.read_csv('heloc_dataset_v1.csv')
```

DATA CLEANING

Special Values

-9, -8 and -7 are special values which represent no bureau record/investigation, no usable/valid trades or inquires and condition not met respectively. In order to get rid of the useless data and avoid noise, we drop the observations that contain -9 and replace the -7 and -8 with nan.

```
# deal with special values
data = data.drop(index = data.loc[(data['ExternalRiskEstimate'] == -9)].index)

for i in range(0, data.shape[0]):
    for j in range(1, 24):
        if data.iloc[i, j] == -7 or data.iloc[i, j] == -8:
            data.iloc[i, j] = np.nan
```

Data Encoding & Processing

In order to make the data set processable, we need to 1) categorize the label values, 2) separate features and label, 3) define DataFrameSelector.

```
# categorize label values
encoder = LabelEncoder()
data['RiskPerformance'] = encoder.fit_transform(data['RiskPerformance'])
```

```
# separate features and label
x_raw = data.drop(['RiskPerformance'], axis = 1)
y_raw = data['RiskPerformance']
```

```
# define DataFrameSelector
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attibute_names):
        self.attibute_names = attibute_names
    def fit(self, X, y=None):
        return(self)
    def transform(self, X):
        return(X[self.attibute_names].values)
```

Missing Values

Since we have replaced -8 and -7 with nan, we now need to replace the nan with mean of that column. In addition, we also standardized the data to offset the impact of data scale on the precision of prediction.

```
# data cleaning: mean for NaN; standardization
feature_names = list(x_raw.columns)
feature_pipeline = Pipeline([
    ('selector',DataFrameSelector(feature_names)),
    ('imputer',Imputer(strategy="mean")),
    ('std_scaler', StandardScaler()),])
x = feature_pipeline.fit_transform(x_raw)
x = pd.DataFrame(x, columns = feature_names)
```

TRAIN MODELS

Performance Metric

In order to compare, we used three performance metics: accuracy, recall and precision.

Decision Tree

The first model we trained is decision tree and we adopt cross validation (cv=10) to get a less biased result. The accuracy, recall and precision we got from decision tree are 0.62267, 0.6121 and 0.612696 respectively.

```
# decision tree
tree_class = DecisionTreeClassifier()
tree_scores_accuracy = np.mean(cross_val_score(tree_class, x, y_raw, scoring='accuracy', cv=10))
tree_scores_recall = np.mean(cross_val_score(tree_class, x, y_raw, scoring='recall', cv=10))
tree_scores_precision = np.mean(cross_val_score(tree_class, x, y_raw, scoring='precision', cv=10))
tree_class_perf = pd.DataFrame({'accuracy': tree_scores_accuracy, 'recall': tree_scores_recall,'precision': tree_scores_
                                columns = ['accuracy', 'recall', 'precision'], index = ['0'])
tree_class_perf
```

	accuracy	recall	precision
0	0.62267	0.6121	0.612696

Random Forest

The second model we trained is random forest and we adopted cross validation (cv=10) to get a less biased result. The accuracy, recall and precision we got from random forest are 0.72468, 0.661738 and 0.726727 respectively.

```
# random forest
rf_class = RandomForestClassifier()
from sklearn.model_selection import GridSearchCV
param_grid_1 = [{'n_estimators':[3,10,30],'max_features':[2,4,6,8]},
                {'bootstrap':[False],'n_estimators':[3,10],'max_features':[2,3,4]}]
grid_search_1 = GridSearchCV(rf_class, param_grid_1, cv=10, scoring='accuracy')
grid_search_1.fit(x, y_raw)
grid_search_1.best_params_
rf_class_best_model = grid_search_1.best_estimator_
rf_scores_accuracy = np.mean(cross_val_score(rf_class_best_model, x, y_raw, scoring='accuracy', cv=10))
rf_scores_recall = np.mean(cross_val_score(rf_class_best_model, x, y_raw, scoring='recall', cv=10))
rf_scores_precision = np.mean(cross_val_score(rf_class_best_model, x, y_raw, scoring='precision', cv=10))
rf_class_perf = pd.DataFrame({'accuracy': rf_scores_accuracy, 'recall': rf_scores_recall, 'precision': rf_scores_precis
                                columns = ['accuracy', 'recall', 'precision'], index = ['0'])
rf_class_perf
```

	accuracy	recall	precision
0	0.72468	0.661738	0.726727

Support Vector Classifier (linear)

The third model we trained is SVC (linear) and we adopted cross validation (cv=5) to get a less biased result. For SVC (linear) model, we tried different values for penalty term C to compare the accuracies. The values of C include 0.001, 0.01, 0.1, 1, 5, 10. From the result, we can know that the model with C = 0.001 has the highest accuracy (0.7312663627669632).

Slide Type

```
# SVC
```

Slide Type

```
clf = svm.SVC(kernel = 'linear', C = 0.001)
np.mean(cross_val_score(clf, x, y_raw, scoring = 'accuracy', cv = 5))
```

```
0.7312663627669632
```

Slide Type

```
clf = svm.SVC(kernel = 'linear', C = 0.01)
svc_scores_accuracy = np.mean(cross_val_score(clf, x, y_raw, scoring = 'accuracy', cv = 5))
svc_scores_accuracy
```

```
0.7299481611599311
```

Slide Type

```
clf = svm.SVC(kernel = 'linear', C = 0.1)
svc_scores_accuracy = np.mean(cross_val_score(clf, x, y_raw, scoring = 'accuracy', cv = 5))
svc_scores_accuracy
```

```
0.7306575347608751
```

Slide Type

```
clf = svm.SVC(kernel = 'linear', C = 1)
np.mean(cross_val_score(clf, x, y_raw, scoring = 'accuracy', cv = 5))
```

```
0.7311652000124461
```

Slide Type

```
clf = svm.SVC(kernel = 'linear', C = 5)
np.mean(cross_val_score(clf, x, y_raw, scoring = 'accuracy', cv = 5))
```

```
0.7309624630636372
```

Slide Type

```
clf = svm.SVC(kernel = 'linear', C = 10)
np.mean(cross_val_score(clf, x, y_raw, scoring = 'accuracy', cv = 5))
```

```
0.7308610945892327
```

KNN

The fourth model we trained is KNN and we adopted cross validation (cv=5) to get a less biased result. For this KNN model, we tried K = 2, 3, 4, 5 to compare the accuracies. From the result, we can know that the model has the highest accuracy which is 0.6885741888914859 when K = 5.

Slide Type

```
# KNN
```

Slide Type

```
clf = neighbors.KNeighborsClassifier(2)
np.mean(cross_val_score(clf, x, y_raw, scoring = 'accuracy', cv = 5))
```

```
0.649733091303886
```

Slide Type

```
clf = neighbors.KNeighborsClassifier(3)
np.mean(cross_val_score(clf, x, y_raw, scoring = 'accuracy', cv = 5))
```

```
0.670117000614331
```

Slide Type

```
clf = neighbors.KNeighborsClassifier(4)
np.mean(cross_val_score(clf, x, y_raw, scoring = 'accuracy', cv = 5))
```

```
0.678128401610478
```

Slide Type

```
clf = neighbors.KNeighborsClassifier(5)
np.mean(cross_val_score(clf, x, y_raw, scoring = 'accuracy', cv = 5))
```

```
0.6885741888914859
```

Logistic Regression

The last model we trained is logistic regression and we adopted cross validation (cv=10) to get a less biased result. The logistic regression model has an accuracy of 0.7307623872233998.

Slide Type

```
# logistic regression
logreg = LogisticRegression()
np.mean(cross_val_score(logreg , x, y_raw, scoring = 'accuracy', cv = 10))
```

```
0.7307623872233998
```

INTERFACE

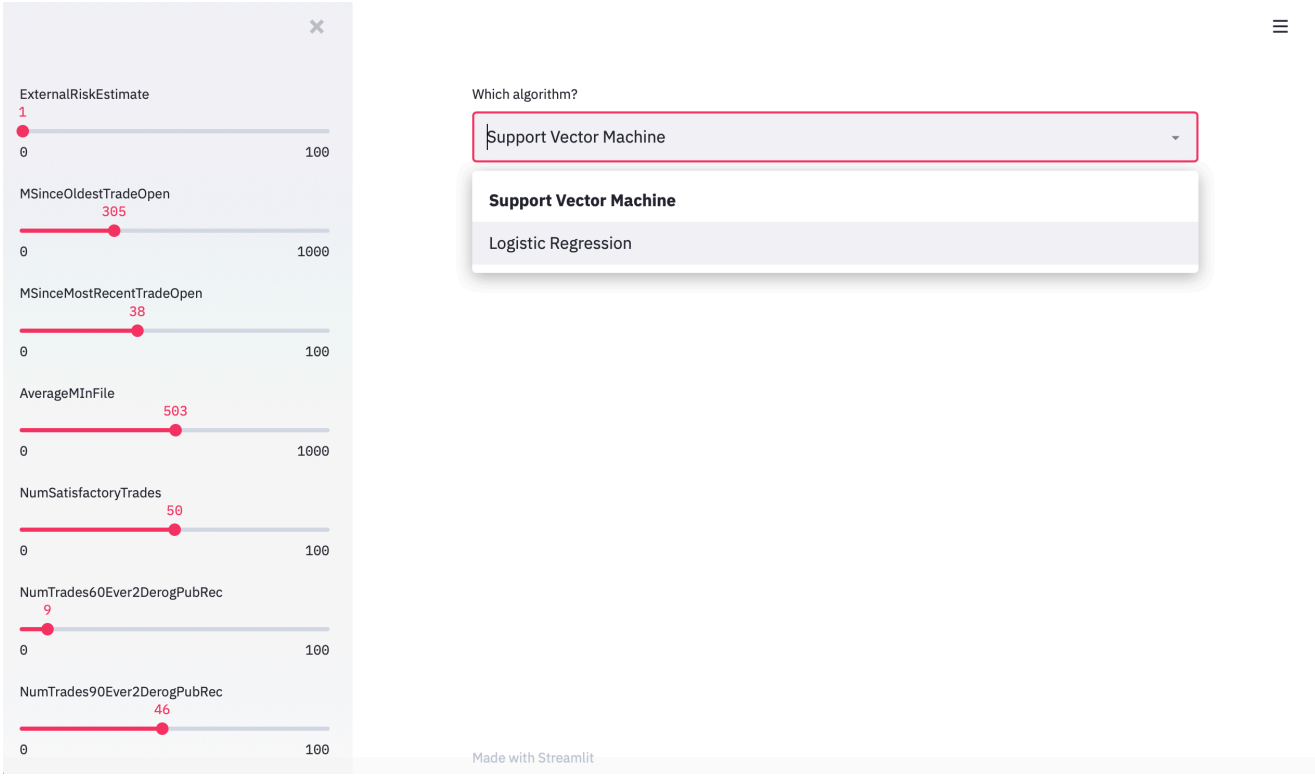
Model Chosen

As we can see from the following table, SVC (linear) and logistic regression outperformed all the other models and their accuracy values are very close. Therefore, we built an interface based on these two models.

	Decision Tree	Random Forest	SVC (C=0.001)	KNN (K=5)	Logistic Regression
Accuracy	0.62267	0.72468	0.731266	0.68857	0.73076

Interface Introduction

The interface we developed using streamlit allows people to see the prediction results generated from support vector classifier and logistic regression model. They can easily see the result 'Good' or 'Bad' by simply changing the value of the parameters.



A/B Test

In order to help users better understand our interface, we would like to explain the A/B test concept we used here. Users would sometimes find that the prediction result changes by changing a single parameter. This means that this single parameter has a significant impact on the prediction result.