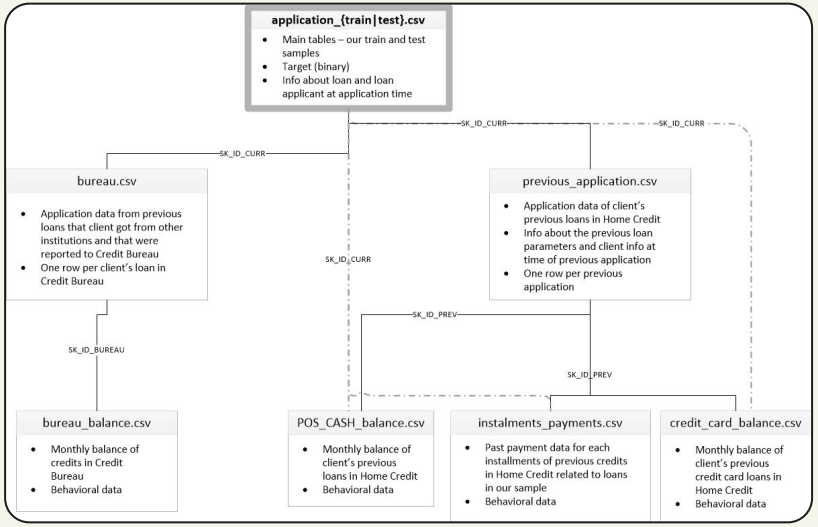
CSC 369

Loan Acceptance Predictor

Anson Yamvinij, Ishaan Sathaye, Michael Montemurno

Our Data

- Home Credit Default Risk dataset from Kaggle
- Loan Applicant as our observational unit
- TARGET: whether an applicant was able to repay the loan without trouble
- 307,511 rows by 122 columns



Our Objective

Play the role of a bank and create a machine learning model to determine if we should give an applicant a loan.

We want our model to be accurate since we don’t want to lose money on defaulted loans.

Preprocessing

1

Units Conversion

- Time variables converted for legibility
- Standardizing features

2

Dimensionality

- Significant reduction in data
- Incomplete applications with missing fields

3

Domain Knowledge

- Relied on domain knowledge to keep features that would be pertinent to model training

Algorithms Implemented

Logistic Regression

With Regularization

K-Nearest Classifier

With Standardization and Euclidean Distance

Naive Bayes

Logistic Regression

- Data split based on training and validation
- Features scaled for normalization
- Gradient Descent
- Class weights added for imbalance
- L2 Regularization for overfitting
- Sigmoid Activation with threshold at 0.5

```
// Train the model using gradient descent
    for (_ <- 1 to maxIterations) {
      val gradients = trainingRDD.map { case (features, label) =>
        val linearCombination = features.zip(weights).map {
            case (f, w) => f * w }.sum
        val prediction = sigmoid(linearCombination)
        val error = prediction - label
        val weight = classWeights(label)
        features.map(_ * error * weight)
      }.reduce((g1, g2) => g1.zip(g2).map { case (x, y) => x + y })

      // Update weights
        val regularization = 0.315
        weights = weights.zip(gradients).map { case (w, g) => w -
            learningRate * g + regularization * w }
    }
```

K-Nearest Classifier

- No real “model”, used a 80/10/10 split for training data, validation data, and testing data
- Features scaled with Standardization
- To avoid too high dimensionality, used only four features per featureset.
- Multiple different configurations tested
  - Different values of K, with ~20 nearest neighbors being ideal
  - Different featuresets, with Amount of income, years employed, age, and amount of credit being the winners
  - Threshold for being considered positive at 0.55

```
def predict(train_data : RDD[(Double, (Double, Double, Double, Double, Double))],
            test_data : RDD[(Double, (Double, Double, Double, Double, Double))],
            k : Int) :  RDD[(Double, Double)]  = {
    // print(k + "\r")
    val revised_data = test_data
                       // do a cartesian product over all of the data
                       .cartesian(train_data)
                       // group by the key of the testing data
                       .groupBy(x => x._1._1)
                       // find the distance for each pair
                       .map(x => (x._1, x._2.map(pair => euclideanDistance(pair._1, pair._2))))
    // output an rdd with each prediction
    revised_data.map(x => (x._1, x._2.toList.sorted.take(k).fold(0.0)((total, n) => total + n )))
                .map(x => (x._1, if(x._2 / k.toDouble < 0.5) 0.0 else 1.0 ))
}
```

Naive Bayes Classifier

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

- 80/20 train/test split
- Standardized features
- 3 Steps
  - Calculate mean and variance for each feature
  - Calculate the probability for each class
  - Calculate Log Likelihood
  - Highest = Prediction

```scala
def predict(model: NaiveBayesModel, loanApplicant: LoanApplicant): Double = {
    val featureValues = Array(
      loanApplicant.NAME_CONTRACT_TYPE,
      loanApplicant.FLAG_OWN_CAR,
      loanApplicant.FLAG_OWN_REALTY,
      loanApplicant.AMT_INCOME_TOTAL,
      loanApplicant.AMT_CREDIT,
      loanApplicant.AMT_ANNUITY,
      loanApplicant.AMT_GOODS_PRICE,
      loanApplicant.REGION_POPULATION_RELATIVE,
      loanApplicant.CNT_FAM_MEMBERS,
      loanApplicant.REGION_RATING_CLIENT,
      loanApplicant.EXT_SOURCE_2,
      loanApplicant.EXT_SOURCE_3,
      loanApplicant.YEARS_BEGINEXPLUATATION_AVG,
      loanApplicant.OBS_60_CNT_SOCIAL_CIRCLE,
      loanApplicant.DEF_60_CNT_SOCIAL_CIRCLE,
      loanApplicant.AMT_REQ_CREDIT_BUREAU_YEAR,
      loanApplicant.AGE,
      loanApplicant.YEARS_EMPLOYED,
      loanApplicant.YEARS_REGISTERED,
      loanApplicant.YEARS_ID_PUBLISH
    )

    val logLikelihoods = model.classProbs.keys.map { targetClass =>
      val classProb = log(model.classProbs(targetClass))
      val featureLikelihoods = featureValues.zipWithIndex.map { case (featureValue, featureIndex) =>
        val (mean, variance) = model.featureProbs(targetClass)(featureIndex)
        val likelihood = (-0.5 * log(2 * Pi * variance)) - (0.5 * pow(featureValue - mean, 2) / variance)
        likelihood
      }
      val logLikelihood = classProb + featureLikelihoods.sum
      (targetClass, logLikelihood)
    }

    val predictedClass = logLikelihoods.maxBy(_._2)._1
    predictedClass
  }
```

Results

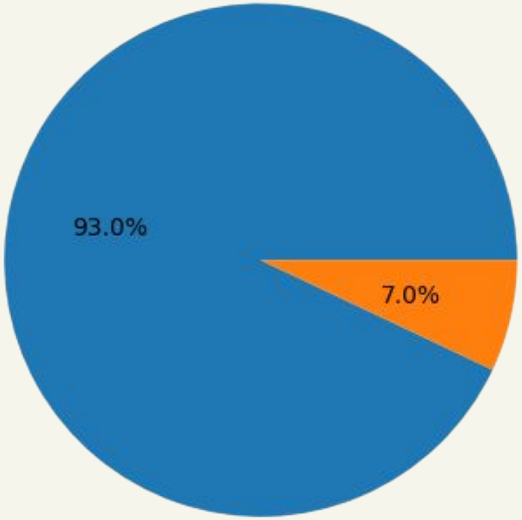
Accuracy, Precision, Recall, F1 Score

Metrics

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.75	0.11	0.29	0.16
KNN	0.90	0.037	0.068	0.024
Naive Bayes	0.876	0.231	0.235	0.233

What we learned

- Working with RDDs and ML algorithms
- Logistic Regression vs Naive Bayes vs K-Nearest Neighbors Classifier
- Impact of Class Imbalance
- Feature Scaling
- Choosing correct metrics



Majority are all other cases of loan statuses vs Minority of loan repayment difficulties

Challenges

Scala Syntax

General challenges that were posed when developing the 3 algorithms

Hyperparameters

Tuning hyperparameters took time and computation power, especially for convergence

Model Selection

Deciding on suitable model among many algorithms required time, experimentation and other multiple factors

Thank you.