Linköping University | Department of Computer and Information Science
Bachelor thesis, 16 ECTS | Programming
2018 | LIU-IDA/LITH-EX-G--18/043--SE

Utilize OCR text to extract receipt data and classify receipts with common Machine Learning algorithms

Använda OCR-text för att extrahera kvittodata och klassificera kvitton med vanliga maskininlärnings algoritmer

Joel Odd
Emil Theologou

Supervisor : Rita Kovordanyi
Examiner : Peter Dalenius

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Utilize OCR text to extract receipt data and classify receipts with common Machine Learning algorithms

Joel Odd
LINKÖPING, Sweden
joel.odd@hotmail.com

Emil Theologou
LINKÖPING, Sweden
emil.theologou@gmail.com

ABSTRACT

This study investigated if it was feasible to use machine learning tools on OCR extracted text data to classify receipts and extract specific data points. Two OCR tools were evaluated, the first was Azure Computer Vision API and the second was Google Drive REST Api, where Google Drive REST Api was the main OCR tool used in the project because of its impressive performance. The classification task mainly tried to predict which of five given categories the receipts belongs to, and also a more challenging task of predicting specific subcategories inside those five larger categories. The data points we where trying to extract was the date of purchase on the receipt and the total price of the transaction. The classification was mainly done with the help of scikit-learn, while the extraction of data points was achieved by a simple custom made N-gram model.

The results were promising with about 94 % cross validation score for classifying receipts based on category with the help of a LinearSVC classifier. Our custom model was successful in 72 % of cases for the price data point while the results for extracting the date was less successful with an accuracy of 50 %, which we still consider very promising given the simplistic nature of the custom model.

Author Keywords

Optical character recognition; Machine learning; Receipts;

INTRODUCTION

Most companies have some sort of system to handle receipts from different types of purchases for the company. Usually the receipts are handled manually and entered into a database. This sequence requires time and that makes it an expensive process. A more effective solution might be to convert this manual process to an automatic one. This could be done by taking an image of the receipt and then let a program automatically extract the information and input that data directly to the database. Extracting the text from the image could be done by utilizing an OCR tool, while creating a program that knows what information it should extract could be done utilizing a machine learning approach.

Machine learning have become a growing field as the collection of large amounts of information have outpaced our ability to make sense of the data. Machine learning come in many different forms and are used for areas where the amount of information is either very large or where the relation between what is evaluated and the result is very hard for a human to quantify.

We used machine learning for data classification, where machine learning can be described as a system that automatically figures out the connections between the features of a training set of test data (a representation of the data to be processed) and a target class of document[1]. Two common ways for this to be returned is as a prediction or as an array of most likely classes with accompanying certainty scores given by the algorithm.

Motivation

The main purpose of this work was to evaluate how a machine learning approach can be used to optimize and automatize the receipt handling process. We wanted to develop a proof of concept for a chain of services from user to database that handles the extraction and classification of receipt data. The main focus was primarily on the preparation of this data to be processed and then extracted with our machine learning solution, and secondly on the extraction of text data from a receipt. By doing this, we wanted to give an insight into how feasible such a system might be at automatically solving a real-world problem.

Research Questions

To get a greater understanding of how machine learning can be utilized for this purpose we have chosen several questions that we will attempt to answer based on the results of this study.

- **Is it possible to correctly classify receipt categories with reasonable accuracy?**
  To get a good understanding of what constitutes a reasonable accuracy, we will use scikit-learn's DummyClassifier and Naive Bayes as a baseline to compare against.
- **To what extent is it possible to extract price and date from OCR text of a receipt?**
  Dates and prices has unique values on each receipt, how feasible is it to try to extract these values with machine learning techniques and what are the specific challenges for this task?

THEORY

This chapter will cover the different techniques that will be used in this thesis.

Optical character recognition

To answer our research questions, we will need a data set of processed receipt text to work on. Optical character recognition (OCR) is a technology that automatically recognizes characters in images and processes this into a text[7], which

enables us to both classify and extract data from them with machine learning models. The five different OCR tools that are relevant for our study are presented below.

Azure Computer Vision API
The first OCR tool chosen for use in our project was from Microsoft Azure Computer Vision API[1]. There is support for uploading images and receive an analysis and is available for free with some restrictions on how many requests can be processed per hour and day.

Google Drive REST Api
The second OCR tool we used was Google Drive REST Api[2]. Google Drive offers the option of opening images with Google Docs[3] and when done in this fashion the image gets saved at the top of the document followed by the extracted text as the inbuilt OCR service interprets the image.

Tesseract
One of the applications we investigated during our work was Tesseract OCR[4]. Tesseract is an open source project that is supported by Google. This is a desktop application where you can select which images to extract text from locally.

Text Fairy
The next application we used was Text Fairy[5], a mobile camera app that extracts the text directly from the image. During research of OCR tools, Text Fairy was found to have good reviews and many recommendations and was therefore chosen for evaluation.

Camscanner
Another tool we wanted to look at was Camscanner[6], a mobile camera app that extracts the text directly when taking the photo with a smartphone. Camscanner offers both a free and premium version. The premium version mentions increased performance for the OCR service as one of its selling points, but only the free version was evaluated.

Data set processing
The source quality is a vital part in the process of collecting data, since if the source contains errors or noise it would affect the results later in the process. Therefore, a solution would be to clean the data in some way. To clean data manually would be time consuming, and an automatic solution would be preferable. However there is no universal solution for the cleaning process but there are different methods that can be used. We present a method[6] with three phases below.

Define and determine error types
The first step is to define what an error is and identify the different types of errors that exist. If a large majority of the data has a general form, then it is likely that a candidate that deviates from this is an error.

[1]https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/
[2]https://developers.google.com/drive/v3/web/about-sdk
[3]https://support.google.com/docs/answer/7068618?hl=enref_topic=2811805
[4]https://opensource.google.com/projects/tesseract
[5]https://github.com/renard314/textfairy
[6]https://www.camscanner.com/

Search and identify error instances
When the errors are defined, the next phase is to identify where the errors are and identify the different variations.

Correct the uncovered errors
The last step is to correct those errors, without corrupting the data set.

Imbalanced data set
Imbalanced data is one of the most common problems in machine learning[9]. If there is a classification category that is relatively rare compared to another classification category it could be considered imbalanced, while another definition is as long as both categories are not approximately equal it is considered to be an imbalanced data set[6]. Other definitions are more strict, and states that when there is a ratio of 1:100 it could be considered imbalanced[2]. As such, there is no absolute definition of when a data set is considered to be imbalanced.

An evident problem is if the data set contains two classification categories with a ratio of 1:99. If a classifier would always guess on the dominant category it would get 99% accuracy, which at first might seem like an impressive result. However, it could be considered worthless because it would never correctly predict the other category[9]. This would give a false impression of the classifiers performance.

Machine learning
Machine learning tries to use intelligent software to get machines to execute their work more efficiently. This intelligence is constructed with learning methods based in statistics and to create these learning algorithms, we need to feed them data to analyze. Because of this, we need a database loaded with any relevant data we can find for the task at hand. Artificial intelligence is often used to describe machine learning, but in reality machine learning is just a small subset of the AI field[8].

In this thesis we will be focusing on the text classification area to answer our first research question, in contrast to applying machine learning with image recognition. In the next section we will give an overview of how algorithms and machine learning work in practice.

Naive Bayes classifier
The Naive Bayes classifier is what is referred to as a probabilistic document classification algorithm. This means that any classification it does will have an associated likelihood of being correct, based on the data the algorithm was trained with.

To produce these probabilities it uses Bayes theorem in its calculations which is where the algorithm gets its name. It also uses a simple assumption that each word's probability is independent of the context, which gives the algorithm its Naive moniker. Despite this simple assumption it has proven to be an effective and competitive tool in several different disciplines[1]. To answer our first research question, we want a baseline to compare against and will use Naive Bayes as part of the comparison.

To simplify the theory behind this classifier, imagine a bag full of words from a text. The words are in disorder but we keep the frequency of how often the individual words occur. We calculate how strongly these frequencies correlate with each type of document we want to classify, and whichever class gets the highest probability is returned as a prediction for the correct class.

Support Vector Machines

A Linear Support Vector Classification (LinearSVC[7]) model is a type of Support Vector Machine. It is mostly used for supervised learning and it is applicable on classification problems. If we have a training set and split this into two categories with a hyperplane, then the hyperplane can split these in many different ways. The goal with the hyperplane is that it want to have a margin as wide as possible to the support vectors. The wider the margin is, the better it can distinguish between the different categories[6].

In scikit-learn LinearSVC works similar to SVC with the kernel="linear" parameter, but has more flexibility[7].

N-grams

Using N-grams is a common method in natural language modeling. An N-gram is sequence of words that are adjacent to each other. A two word sequence could be "my name" (bigram) and a three word sequence could be "my name is" (trigram). When using an N-gram it is possible to calculate the probability of seeing the first word given the last and vice versa[4]. This is used both by our custom made model to answer our second research question and also in scikit-learn's vectorizer function before processing the text into vectors.

Custom N-gram model

The model is a trigram model that builds a vocabulary of all trigrams present in the data set containing the data point that we are searching for as the middle token. The vocabulary consists of three parts, each keeping track of the different positions of tokens. These positions are the token left in the trigram, the token right in the trigram and also a tuple of both the left and right token.

The vocabulary is used to create a dictionary of probabilities, where each position is mapped against the likelihood of a particular token occurring at this position in relation to a data point we are searching for. As such, the probability is the occurrences next to our data point divided by the total amount of occurrences.

When making a prediction, the model parses through each trigram in the document it is guessing on while comparing the left and right tokens in the trigram against the dictionary of probabilities. Whichever trigram has the highest probability score is considered to be most likely to contain the desired data point in the middle position, and the middle token is returned as the answer.

Neural networks

Neural networks are inspired by theories of how biological neurons in the brain function. They are based on the idea of a propagation network whose decisions are made depending on how activation thresholds trigger, which propagate to a result layer. In the the result layer, the node with the highest activation value decides the network's answer[1].

For the basic perceptron model, we are using constructs known as perceptrons which the model gets its name from. The perceptron is a very simple type of neuron, that triggers based on weighted inputs complemented with a bias value which gets added to the total value for the perceptron. Simplified, the perceptron triggers if the sum of the weighted inputs and bias exceeds a threshold value. In practice, we usually have three layers in what is referred to as a multilayered perceptron model or a multilayered neural network. This consists of the input layer, the hidden layer and the output layer, where the hidden layer make the calculations and the input layer only accepts the starting state of the query. The input layer is a representation of the data we wish to process and the hidden layer is the representation of how the system interprets connections between the data which contains the logic of the model. Finally, the output layer is the where we can retrieve the system's answer. The hidden layer may be composed of several internal layer's of perceptrons chained together[1].

When training the network, we tweak the weights and thresholds in the hidden layer with the help of training algorithms to maximize the certainty of the output layer's prediction[1]. By doing this incrementally for our training data we build up a configuration in the hidden layer that is adept at analyzing the data we feed into the system. This allows us to set up a very complicated network of dependencies and assumptions of the data automatically.

Scikit-learn

Scikit-learn is a free open source and commercially usable machine learning tool for data analysis, used by well-known companies such as Spotify, booking.com and OkCupid[8]. It has many classifications models included[7] which can be used to classify receipts.

CountVectorizer

The vectorization and tokenization is handled by scikit-learn's CountVectorizer[7] module. The vectorizer module provides several handy operations to be performed on the text, such as a tokenizer that can divide the text string of the documents into a number of tokens selected by a match of a regular expression that can be replaced. It also provides the option of choosing what range of N-grams the document shall be processed into. These are provided by specifying what the smallest N-gram is desired to be, where a value of 1 represents a unigram, up to the desired value. The CountVectorizer takes all tokens and creates a matrix vector with the tokens based on how often they appears.

TfidfTransformer

TfidfTransformer[7] is a built-in function in scikit-learn. It takes the previous CountVectorizer count matrix and normalizes it to a tf or a tf-idf representation. Basically, it is a way of optimizing the vector space and changing the impact of

[7]http://scikit-learn.org/stable/modules/classes.html

[8]http://scikit-learn.org/stable/testimonials/testimonials.html

specific words depending on how they appear in the document. As one example, a word that occurs very frequently across all classes tells us very little when we find this word in a new document and the significance of this word is therefore reduced. It is presented in more detail in the scikit-learn documentation[7].

Cross validation metric

The process of cross validation is equivalent to shuffling the data, dividing it into N equally sized parts and then training the algorithm on N-1 of these parts and using the last part as test data to compare against. The training and testing is then repeated while choosing a different part as the test data, and using the rest as training data until the entire data set has been used as test data once. The result for each part of test data (fold) is then returned as an array that can be averaged to get a single accuracy metric.

The advantage of this metric is when some parts of the data set are easier or harder to classify than other parts. As we move the training set over the entire data set, we reduce the risk of getting a misleading view of the performance of the classifier if that is the case. In this work we are using scikit-learn's cross_val_score[7] module for this functionality.

DummyClassifier

The DummyClassifier[9] is a simple model not meant for real problems, and follows simple rules such as always guessing on the most common class (maximum likelihood estimation) or doing random guesses based on the distribution of the data (stratified).

Confusion matrix

To describe the performance of a classification model on a data set, it is possible to use a confusion matrix. A confusion matrix offers the ability to summarize the predictions for every category by mapping the actual labels on the data against the predicted values returned by the model in a two-dimensional table, where correctly predicted results are presented in the diagonal[3]. In this work we are using scikit-learn's confusion_matrix[7] module.

Gridsearch Optimization

Scikit-learn offers GridSearchCV[7] which performs an exhaustive search on a dictionary of parameter values, where each key maps to a parameter and each value maps to a list of parameter values to test. This then performs a cross validation test with three different folds of the data as default and outputs the configuration that scored the highest. This means the exhaustive search need to retrain the model on three different configurations of data per configuration of parameters, which means it grows in complexity exponentially fast. For a setup with two parameters with four possible values each, we would need to train the model (3*4*4) times, i.e. 48 times. The cross validation score is used to compare every configuration against each other and the top scoring configuration is returned as the result.

[9]http://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html

Related work

While the research for machine learning is a growing field, there are few research papers concerning the specific task of analyzing and classifying receipt-data. The most relevant study[5] showed promising results using the J48 and KSTAR algorithms for this task. However, they note more research is needed for credible results and their work relied on manual retyping of the receipt with no automatic datafication of the physical receipts.

This work also included a very small specific subset of receipts for analysis, while we will need to broaden the scope to more diverse sources of data. Still, the results in this study give an optimistic view of the feasibility of the machine learning approach for the classification of the receipts, given correctly classified base-data.

Delimitations

We chose only to use OCR and text extraction. We are choosing to use scikit-learn's machine learning library. The data points chosen is only the total price and date of purchase.

METHOD

This chapter will describe the process leading up to being able to classify and extract information from receipts using machine learning. The process went from collecting the physical receipts, transforming them to digital data, selecting and training machine learning algorithms on that data, writing a custom made algorithm to extract specific fields of data from the receipts that we are interested in and finally evaluating the classifiers performance. See Figure 1 for an overview on the work flow.

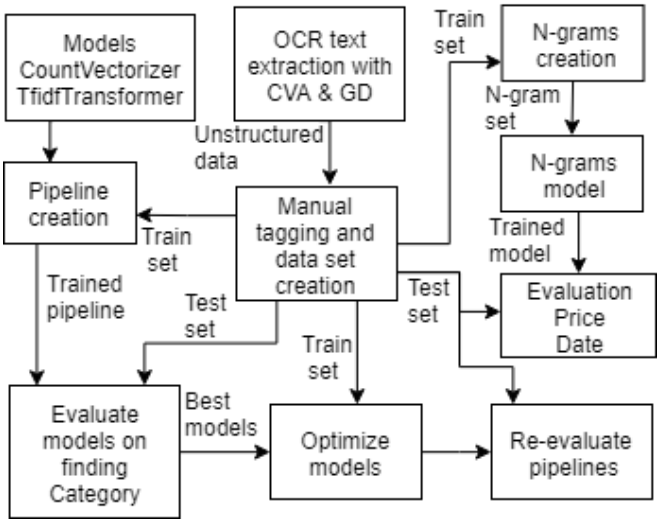

Figure 1: The flowchart shows the process from receipt image to comparing the results

Receipt

The data set we are using for this project is 556 receipts. Those receipts are divided into 5 major categories, grocery stores, restaurants, retail shops, travel-related and miscellaneous (see

Table 6 in Appendix). All the receipts are from Swedish businesses and almost all are less than a year old. Grocery stores, restaurants and retail shops are the biggest major categories with the least amount of variance while travel-related and miscellaneous are smaller and have more mixed types of receipts. Variance in this context refers to the fact that a parking receipt and a gas station receipt both reside in the travel-related category and will have very different formatting, length and keywords and as such will have much less variance than two receipts from different grocery stores compared to each other.

The distinction "major category" is made to differentiate between the other categorization of subsets within the major categories, referred to as "sub categories". The sub categories were created by dividing each major category in smaller subsets, most commonly by a particular company that represented a significant part of the major category. Any receipt not fitting into any of these subsets was categorized as the sub category other that was a sub category present for every major category.

We expected the variance in the two sets travel-related and miscellaneous to result in a worse classification score for our classifier. We also expected that the total number of receipts in each category to impact the performance. This means we expected our classifier to achieve a better score for the larger set miscellaneous than travel-related, even though both sets have a similar level of variance in the design. Conversely, we expected the major category grocery stores to perform the best, as this is the largest major category with the least amount of variance. The metric "variance" is mostly our own subjective interpretation in this context, an interpretation made after manually parsing every receipt when producing the metadata.

OCR

To be able to analyze the data on the receipts, we need to convert them into a digital form that can be processed by the machine learning algorithms. There were several different approaches tested with varying degrees of success and most of the evaluation was done by testing different techniques that we could find recommendations for.

These included Tesseract, Text Fairy, CamScanner, Google Drive REST Api and Azure Computer Vision API.

Prestudy

The benchmark was done in two parts, where the first prestudy was a more informal test where the apps where tested on a few receipts to investigate if they were suited for the task. If we felt doubtful the OCR tool wouldn't be able to reach at least 25 % accuracy in later tests, the solution was deemed not promising enough to warrant a full benchmark.

Benchmark

Image quality

Image quality is a parameter that we need to research if it has any impact to the OCR. For this we chose three types of settings for the photo session to benchmark the OCR tools that passed the prestudy.

1. Environment 1 was photographed with a smartphone camera in a normal office lightning environment. The images were taken with a Sony Xperia Z5 Premium in a smaller room with two ceiling lights with the receipt lying flat on a table. This represents a normal photo picture as we expect an end user might take the photo if this was part of an application. As we held the mobile phone above the receipt when the image was taken and the direction of light is from above the camera, there is a small amount of shadows on the receipt from the phone when taking the picture. The photographer positioned himself at the midway point between both lights to minimize this effect.
2. Environment 2 was represented by a smartphone camera photo with a small spotlight as an extra light source. The same mobile camera as above was used. The spotlight both gives brighter lightning and does not cast any shadows on the receipt. The receipt was also placed on a flat surface on top of a white piece of paper to minimize interfering noise from the background. Two small pieces of metal was placed at the top and bottom edge of the receipt to flatten and stretch the receipt out as much as possible. This was to represent the best case scenario for the receipts.
3. Environment 3 was similar to environment 2, but the smartphone camera was exchanged with a Canon EOS 450D system camera to get the best possible opportunity for the OCR algorithm to extract the receipt text.

Image evaluation

To evaluate which photo environment that we would continue with, we used a small sample size of 10 receipts. The size of the receipts was one of the factors we expected to influence the effectiveness of the result from the OCR algorithms the most, so we chose 4 small, 3 medium and 4 large receipts for the test set. The difference in size between each category is about 5 centimeters increase when moving to a larger subset. We also avoided using receipts from the same store, with one exception. Two receipts was from the same store, one in the small category and one in the medium since this particular store had a problem with lower quality print of receipts. As such, we deemed it a good candidate for evaluation to see if the improved photo environment lessened the amount of errors from the OCR algorithm. With these restraints in mind, we picked the rest of the testing set randomly from eligible receipts.

To have a baseline to compare against, we manually parsed all ten receipts and made a text document for each receipt that would represent the output we would expect from a theoretically perfect OCR algorithm. This document was then tokenized by simply splitting the text string at every white space and the tokens compared against the tokens in the document from the OCR algorithm. This manual set will also be referred to as the golden standard text.

Data set

To construct the data set, all the images were divided according to the categories we felt that they belonged to and as such are subjective interpretations. Some of these were not as easy to classify as others, for example the IKEA receipts were deemed to belong to the retail shop category but IKEA also has a separate in-house restaurant.

We decided that the main identity of the particular instance of store should signify the category, and therefore the IKEA receipts from the restaurant were labeled as restaurant receipts as the restaurant inside IKEA was counted as its own establishment. Another issue was larger grocery stores which offers a wide selection of items for sale, including entire sections of the store that sells items more reminiscent of a retail store. Following the rule clarified above, receipts from grocery stores with no sale of actual groceries were still counted in the grocery store category.

We had two data sets, one for each OCR algorithm that passed the prestudy (see Results). Both data sets were constructed by using images from environment 1, defined in the Image Quality section above (see Discussion for motivation). The results returned from the OCR processing was saved as a document string in a csv-file[10] under the label "Tokens", as one of seven categories in the csv-file. The other six contained metadata about the receipt; what major category the receipt belonged to, what sub category, the date of purchase, the total price of the transaction, a string representing the tax rate and the filename of the image that originally was processed by the OCR algorithm. The tax rate field was never used (see Discussion).

While the "Tokens" field was automatically extracted in a process described more in-depth in the next section, the other six categories were all manually tagged for each receipt by going through all receipts individually. Some issues when tagging the receipts included receipts with the date written in several different formats. Since most dates are written in one of several common places on most receipts, the first problem was solved by being as consequential as possible when choosing which date position takes precedence.

To account for common errors in the OCR algorithm, there was a small amount of processing of the OCR text before saving it in the csv-file. This was done both to give a more fair result when comparing the OCR text against the golden standard text and also to give the classifiers more consistent training data. There were two different types of data that were processed. One was exchanging any comma in a price into a period (i.e. "19,90" -> "19.90"). Another part was the issue of a particular type of metadata present on many receipts, for example AID number, ref number, tsi and tvr number among others. These field are usually in the form of "<keyword>:<number>". However, the OCR algorithms proved to be somewhat inconsequential about whether to interpret the space between the colon mark and the number field as white space or not. We decided to split these fields up in the script that constructed the csv-file with two simple regular expressions, and always insert a space after the colon if there was none[11].

Token field extraction

To extract the tokens, one solution for each OCR tool was developed. For Azure Computer Vision API, we built a python script that calls their API with a request an image. The Azure framework then returns a JSON-file[12] that contains the text information which we can extract with a script. The JSON-file is returned with a lot of meta information about each piece of text, such as bounding boxes containing what coordinates in the image each piece of text was discovered at. The metadata was stripped away and the processed text was saved as a string in a csv-file meant for use to train our algorithms.

Google Drive offers the option of opening images as documents and when done in this fashion the image gets saved at the top of the document followed by the extracted text as the inbuilt OCR service interprets the image. Using their provided API, we built a python script to connect and upload our receipt images one at a time, save them as a documents and then requesting a simple text file of the document as response. When requesting a text file as response, formatting such as text size, color and the image itself gets stripped away so we can attain only the actual text data we are interested in. When we have received the result, our script requests that the document on Google Drive that was created as an intermediate step in the process gets deleted.

Data preparation

Before feeding the data to any algorithm, both in the benchmark, optimization or the regular classification training, there are some steps that need to be made to prepare the data by transforming it in a way that makes the training and prediction more optimized.

The main steps of this procedure is the act of vectorization and tokenization followed by the act of tfidf-transforming the data. This step was included as part of a scikit-learn pipeline[7] in our solution for practical reasons, but this can also be included as a separate entity or step. There is also the option of using the built-in scikit-learn function TfidfVectorizer[7], which combines both the CountVectorizer and TfidfTransformer into a single step, a solution that is equivalent to calling both after each other. We chose to separate these modules to get better control during the optimization and kept it this way during other training steps to keep the training and optimization as similar as possible.

Classifier selection

To choose what kind of classification models that were suited for our task, we followed the advice from two different cheat sheets. Since we were working with scikit-learn's library and also with Azure services, we used resources provided by them as a guideline as we expected that the advice would match the particular implementation of the classifiers. We chose the first guideline from Microsoft Azure Machine Learning[13]. The second guideline is from scikit-learn[14].

Based on these documents, we chose to use the following classifications models for evaluation: the support vector machine SVC[7] with rbf and linear kernel, LinearSVC[7], Naive Bayes model MultinomialNB[7],

[10] https://tools.ietf.org/html/rfc4180

[11] https://docs.python.org/3/library/re.html

[12] http://www.json.org/

[13] https://blogs.msdn.microsoft.com/uk_faculty_connection/2016/12/07/azure-machine-learning-cheat-sheet/

[14] http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

LogisticRegression[7], multi-layered perceptron model MLPClassifier[7], KNeighborsClassifier[7], and last the ensemble classifiers, Randomforest[7], AdaBoostClassifier[7], ExtraTreesClassifier[7] and the DecisionTreeClassifier[7].

These algorithms were selected for a larger first batch of benchmarks without tuning their hyperparameters, to get a general sense of the algorithms fit with our data. We then proceeded with the best of these models for hyperparameter tuning and optimization before proceeding with a second round of benchmarking.

Evaluation

The benchmark was done by training each classifier on the data set provided from Google Drive REST Api and measuring the performance of the classifiers ability to correctly predict which major category the receipt belonged to.

The evaluation was done with a cross validation metric. To be able to get comparable results over different runs and when comparing different algorithms, the same random seed was used for the process of shuffling the data.

The benchmarking program was a simple python script that made a pipeline with a CountVectorizer and TfidfTransformer and added a new classifier to the pipeline for a round of testing against the data set. The classifier in the pipeline was then replaced with the next classifier, and the process repeated for every classifier.

Optimization

The optimization was performed with scikit-learn's GridSearchCV algorithm. After adding different parameters for the CountVectorizer, TfidfTransformer and the most impactful parameters of the classifier, we ended up with about 4000 different configurations to test for LinearSVC.

Because the MLPClassifier takes a lot longer to train than the other classifier chosen for the second round of benchmarking, we needed to customize the optimization, which instead was performed in batches. As the MLPClassifier takes about 30 seconds to train on our machines for each configuration, we instead opted to only modify a few parameters at once. Each batch consisted of about 80-100 configurations that were then cross-examined and 10 batches was tested in total. This means the MLPClassifier has the highest likelihood of being farthest away from its optimal parameters.

Evaluation

The process was similar as the benchmark done before the optimization, but the pipelines was tailored specifically to each classifier. The CountVectorizer, TfidfTransformer and classifier all were fitted with the optimal hyper parameters decided by the GridSearchCV algorithm (see Figure 4 in Appendix for configurations) and each pipeline was evaluated with cross validation separately.

We have also included two simple classifiers to compare against. The DummyClassifier, using the stratified prediction parameter, was included based on advice from scikit-learn's documentation, recommending this as a quick sanity check. We have also included the Naive Bayes classifier as this is commonly used as a baseline[10].

For the top scoring algorithm we produced a confusion matrix, which was done on a single fold of data instead of using cross validation since this better suits the presentation format of the confusion matrix.

Data point extraction

We also wanted to research if we could use machine learning to extract specific data points from the receipt. When referring to a data point in this context it is considered to be a specific type of token containing a value that changes between receipts, in our case total price and date of purchase. In our solution, the token may contain different formatting or style in different documents, the only needed restriction is that it is a singular token which makes it suitable for extracting data points in most cases.

The process of extracting specific data points is very different from classifying the receipts. As such, we could not use the same algorithms or classifiers for this task that was used previously in this work. We also did not wish to vectorize or process the tokens as the results needed to be returned as plain text when the correct tokens were identified, and the vectorization process does not offer enough features to our solution to justify the added complexity of returning the plain text tokens after the data set already has been processed.

Because of this, we decided to construct a custom N-gram model as a classifier and a proof of concept. We hoped that this would give us a greater insight into the challenges of how receipt data is designed and also give us a greater degree of control in how the data is extracted.

As the model only considers tokens surrounding the data point and not the data point itself, it can be exchanged for any other point of data as long as the data is represented by a single token. As such, it is not translatable to be able to extract data divided into several tokens and accuracy will deteriorate the less consequential the surrounding tokens are between different documents.

Evaluation

The data set is divided into a training set and a test set. The training set consist of 80 % of the documents and the test set is the remaining 20 %. Both sets are divided in two parts, one part containing the receipt text as extracted by the OCR and the other one containing the value for the data point we are looking for, referred to as the label. When training the algorithm the model has access to the labels to know what it is looking for while the prediction function only has access to the document text. The labels for the test data is instead used after the predictions have been made by the model to compare against.

RESULTS

OCR results

The results of the OCR comparison showed the choice of OCR service had the biggest impact, while different photo environments were much less impactful. The different photo

environments was described in the previous chapter, and will in this chapter be referred to by the numbers associated with each option as they were introduced. To reiterate, environment 1 was a smartphone camera with no extra lighting, environment 2 was using a small spotlight on a flat white surface using small metal weights to flatten out and stretch the receipt. Environment 3 is the same as environment 2, but with the smartphone camera replaced with a system camera.

Prestudy

The tools deemed acceptable and practical were OCR through Google Drive REST Api and Azure Computer Vision API. The others had such poor results that we felt confident more rigorous benchmarking of these tools would be superfluous.

Benchmark

The accuracy metric of the receipts against the gold standard is computed as the number of correct tokens divided by the number of total tokens. The total amount of tokens is the amount of tokens in the document created by the OCR algorithm and not the number of tokens in the gold standard document. This is because noise provided by extra inaccurate tokens created by the OCR algorithm is seen as unwanted.

The results show Google Drive REST Api performed significantly better, where Azure Computer Vision Api had a problem with over-segmentation of a lot of tokens (splitting words into several tokens in unwanted places). This was a common theme for all three types of images, regardless of setting and quality of the photo. Neither solution showed any significant improvement from using pictures taken under a dedicated light-source or with a high-fidelity system camera. The results of this benchmark is presented in Table 1.

Table 1: Table shows the accuracy results from the OCR benchmark.

Environment	Google Drive REST Api	Azure Computer Vision Api
1	87.16 %	53.99 %
2	84.12 %	55.80 %
3	88.79 %	52.55 %

Classification results

This section will provide the results from the classification task of assigning the correct category to each receipt.

The classification performance metric is the mean of the result from doing a cross validation evaluation on the data set with three folds. When referencing the accuracy of the algorithms moving forward, it is the mean value of this cross validation that it is referring to.

Initial classifier benchmark

This was run against the Google Drive data set. Figure 2 gives an overview of the evaluation of every classifier in the non-optimized benchmark.

The top two performing classifiers that was chosen for optimization and further evaluation are presented in more detail in table 2.

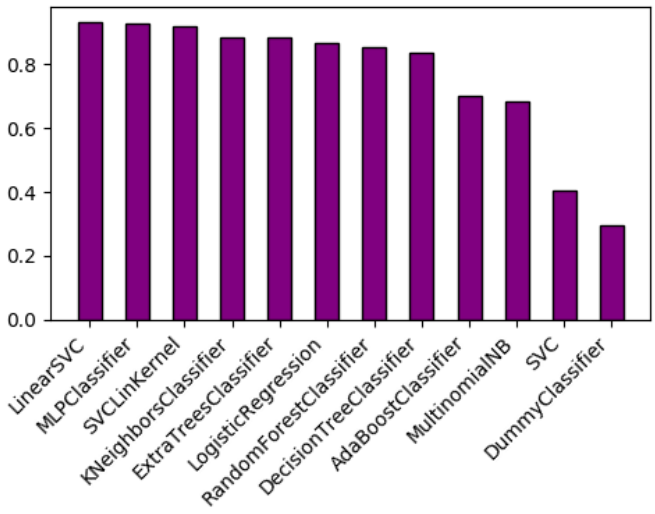

Figure 2: Accuracy on the label "Category" with different algorithms on the Google Drive data set. This result decided what algorithm we chose to optimize.

Table 2: Table shows GD results for the top 2 classifiers LinearSVC and MLPClassifier on the major categories.

Algorithm	default
LinearSVC	93.16 %
MLP	92.98 %

Table 3: Results for the top 2 algorithms on major categories post-optimization and the difference in accuracy compared to pre-optimization. Two baseline algorithms are included as reference to LinearSVC's accuracy.

Algorithm	Optimized	difference
LinearSVC	94.07 %	+ 0.91
MLP	93.53 %	+ 0.55
Naive Bayes	68.52 %	- 25.55
DummyClassifier	28.06 %	- 66.01

Table 4: Accuracy on sub categories. The difference in this table is compared to the accuracy of the major category predictions.

Algorithm	Sub category	difference
LinearSVC	92.27 %	- 1.8
MLP	90.29 %	- 3.24

Optimized algorithm results

After running the optimization search we managed to improve the accuracy by a small margin. The default values were found to be very competitive and the difference in accuracy between the optimized algorithms only translate to an extra 2-5 documents classified correctly. In the case of Naive Bayes and the Dummy classifier, the difference field references the difference against the top classifier LinearSVC. The results for the accuracy is presented in table 3.

The confusion matrix for LinearSVC is presented in figure 3. These values are calculated on a single fold of data with 65 % as training data and 35 % as test data. We also tested LinearSVC and the MLP classifier on the smaller sub categories and the results are presented in table 4.

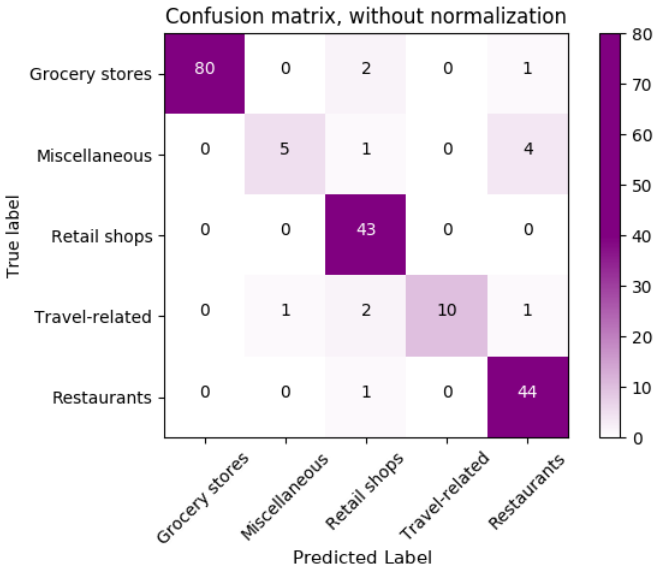

Figure 3: The confusion matrix for the LinearSVC classifier after optimization, performed on the Google Drive data set.

Accuracy on extracting keyword values

As this task is very binary in nature, where either the model finds the data point or not, accuracy is the only metric we are interested in. See table 5 for the results.

Table 5: Table shows the accuracy for Price and Date.

Data point	Accuracy
Price	72.32 %
Date	50.89 %

DISCUSSION

Results

One of the first things we evaluated was the image quality, we believed the environment the photograph was taken in would have a large impact on the outcome of the OCR. It seems it did not have the impact we anticipated, and trying to improve the situation sometimes made it worse. Our conclusion is that modern smartphone cameras are good enough for this to not be as important. However, our evaluation of this was on a very small subset of photographs and may not be translatable to more photos or other devices, and the results would need to be performed in a much larger context in a more controlled environment to get verifiable results.

The results for the OCR benchmark was rather inconclusive, with each environment having very similar scores. Based on this, it was decided we should use environment 1 to construct our data set, as this both represents the expected setting a picture would be delivered to our application from a potential user and the results from this approach showed satisfactory performance.

Overall the classifiers performed reasonably well on the categories where we expected sub-optimal performance, for example on the miscellaneous and travel-related major categories which both are very varied, with receipts from a wide range of stores and very different formats. Also, only dropping a few points of accuracy on the cross-validation tests when comparing against the sub categories was unexpected, as comparing on both a greater range and smaller subsets of categories increase the complexity of the prediction while simultaneously diminishing the training data available per subset.

However, we did not see much improvement from tuning the hyperparameters. One contributing factor to this might be that is difficult to achieve much higher accuracy, as when inspecting the miss-classified receipts they were often outliers in terms of layout, had bad print, were very short or a combination of these. To improve the accuracy further, the most cost-effective way in a a future solution might include an OCR-solution custom-made for the task and a larger population of receipts to train on.

When comparing the results of the data point extraction, a baseline consisting of a parser retrieving the data with regular expressions was discussed but was deemed troublesome. The complexity of the regular expression can be made extremely complicated and the amount of complexity chosen would have been arbitrary, and we therefore decided using this as a baseline for comparison was not a good solution.

Initially the date of purchase, the total price and the tax on the receipt was chosen for investigation. However, many receipts had several different tax rates and the standard for how to print out the information differed between different stores. Because of this, to be able to extract the tax-rate we would need a custom solution with a much greater complexity than the other two fields, and this was deemed out of scope for this work.

Between the remaining data points, the date data point proved to be the most difficult to work with, for a couple of different reasons that needs to be solved in future systems. The solution we have implemented can not handle data points containing white space because of our tokenizer, however some stores use date formats containing spaces which means we get a 0 % accuracy for these sub categories, for example receipts from the Netto chain of stores. There is also the problem with the OCR algorithm not being able to parse the date in the first place, which occurred in about 10 % of receipts for the date field. Lastly, another problem was the fact that the date could be printed in different formats and multiple times on the same receipt which makes the training and tagging of data more difficult. Most of these problems were present for the price data point as well, but much more seldom.

A solution with a more sophisticated tokenizer that utilizes processing of the features in the data set might see much more impressive results. Also, these results can be improved very easily by including some simple regular expressions to filter the possible predictions. This was however deemed out of scope for this study as we desired generalized solutions. Us-

ing word shape features[15] might also be a powerful tool to use to process the text to improve the models results while still producing a more generalized model than can be achieved by using regular expressions. A data set where every word has been tagged with different classes also opens many possibilities of constructing more powerful models but would demand a significant amount of manual work to produce.

Error sources

A possible source of error could be our definition on how to structure and create the different categories. At present, we are dividing them by classes that are to us semantically similar and as such it is a subjective division. It is possible that a separate algorithm can be used to automatically cluster the data into sections and if used as a base for constructing and dividing the categories, this might yield better results.

Increasing the data set size would also give more indicative results, as a data set size of only 556 receipts means many categories are quite small and might not yield enough training or test data. Also depending on the system implemented, the desired population of receipts might be very different with receipts instead from different types of companies or from different countries than included in this work. However, as our work is only a proof of concept we think the results are clear enough to give a good understanding of how suitable machine learning is for this problem even if the categories are not optimally divided.

Another possible problem with our solution is how well these results would map to unseen data. There might be a concern that the model learns to recognize data points such as company names or addresses and connects these to the correct category. As such, the model might have a much worse accuracy when used on new receipts from companies or cities it has no data from during training. However, the sub category other under the restaurant major category contains very varied receipts from different companies and cities with many companies only appearing a single time (see Table 7 in Appendix). If the previous concern was correct, our accuracy would drop sharply when predicting on this subcategory. However, accuracy for this category is still in line with other categories and as such we believe our classifier is still suited to predict on new data as long as it it reasonably compatible with the data in the training set. Still a larger study on a more varied population on receipts would give more verifiable results.

Conclusion

Is it possible to correctly classify receipt categories with reasonable accuracy?

With around 94% accuracy we deem this as very reasonable while keeping the somewhat small data set evaluated in mind. Sub categories which are much smaller subsets with less differences between categories still got a very high accuracy with the evaluated model. There were many classifiers that managed to score 80 % or more in the cross-validation benchmark which might signify that classifying receipts is a somewhat easy task that machine learning is very suited for to begin with, especially since the hyper parameter optimization did not see significant improvements and compared to the Naive Bayes and Dummy classifier we outperformed them with a great margin. We think these results are very promising, but as these results only stem from a proof of concept solution, a larger study might be needed to verify them.

To what extent is it possible to extract price and date from OCR text of a receipt?

We got 72.32% accuracy on retrieving the price and 50.89% accuracy on extracting the date. However, this was done with a simple custom made solution that has great potential for improvement and more sophisticated solutions are expected to perform much better. As such, we deem these results as very promising and they might warrant time and research spent on evaluating a more complex solution.

[15] https://web.stanford.edu/ jurafsky/slp3/21.pdf

REFERENCES

1. C.C. Aggarwal. 2015. *Data Classification: Algorithms and Applications*. CRC Press. https://books.google.se/books?id=nwQZCwAAQBAJ
2. Haibo He and Yunqian Ma. 2013. *Imbalanced Datasets: From Sampling to Classifiers*. Wiley-IEEE Press, 216–. DOI:http://dx.doi.org/10.1002/9781118646106.ch3
3. G. James, D. Witten, T. Hastie, and R. Tibshirani. 2013. *An Introduction to Statistical Learning: with Applications in R*. Springer New York. https://books.google.se/books?id=qcI_AAAAQBAJ
4. D. Jurafsky and J.H. Martin. 2009. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson Prentice Hall. https://books.google.se/books?id=fZmj5UNK8AQC
5. M. Karlovčec. 2011. Information extraction from receipts using machine learning. In *Proceedings of the ITI 2011, 33rd International Conference on Information Technology Interfaces*. 477–480.
6. O.Z. Maimon and L. Rokach. 2005. *Data Mining and Knowledge Discovery Handbook*. Springer. https://books.google.se/books?id=tdNQAAAAMAAJ
7. Ravina Mithe, Supriya Indalkar, and Nilam Divekar. 2013. Optical character recognition. *International journal of recent technology and engineering (IJRTE)* 2, 1 (2013), 72–75.
8. M. Mohammed, M.B. Khan, and E.B.M. Bashier. 2016. *Machine Learning: Algorithms and Applications*. CRC Press. https://books.google.se/books?id=X8LBDAAAQBAJ
9. Hoens T. Ryan and Chawla Nitesh V. 2013. *Imbalanced Datasets: From Sampling to Classifiers*. Wiley-Blackwell, Chapter 3, 43–59.
10. Sida Wang and Christopher D Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*. Association for Computational Linguistics, 90–94.

APPENDIX

Table 6: Receipt categories and amount.

Major category	Sub category	Amount
Grocery stores		
	Drugstores	9
	Coop	28
	Ica	82
	Hemköp	54
	Netto	16
	Willys	24
	Other	13
Retail shops		
	Akademibokhandeln	15
	Ikea	13
	Clothing stores	20
	ÖoB	11
	Other	75
Travel-related		
	Parking	12
	City Transportation	10
	Gas station	10
	Other	2
Miscellaneous		
	Other	24
Restaurant		
	Chili lime	13
	Lai thai	41
	Other	84

Table 7: Restaurants sub category Others. The table presents the number of occurences for each unique business. A business is a particular restaurant or chain of restaurants.

Amount with same category	Amount occurrences
Count of 1	38
Count of 2	8
Count of 3	2
Count of 4	6

```
pipelineLinSVC = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1, 1),
                             analyzer='word',
                             token_pattern=r'\b\w\S+\b',
                             decode_error='strict')),

    ('tfidf', TfidfTransformer(use_idf=True,
                               norm='l2')),

    ('clf', LinearSVC(max_iter=500,
                      C=1,
                      tol=0.0001,
                      fit_intercept=True,
                      multi_class='crammer_singer'))
])

pipelineMLP = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1, 1),
                             token_pattern=r'\b\w\S+\b',
                             decode_error='strict')),

    ('tfidf',  TfidfTransformer(use_idf=True, norm='l2')),

    ('clf', MLPClassifier(solver='adam',
                          alpha=0.01,
                          hidden_layer_sizes=(210,),
                          activation='relu',
                          learning_rate='invscaling'))
])

pipelineMNB = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB())
])

pipelineDummy = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', DummyClassifier())
])
```

Figure 4: Optimized hyperparameters presented in their pipelines