November 2018

Deep Learning for Data Privacy Classification

Griffin R. Bishop
Worcester Polytechnic Institute

Harutyun Sadoyan
Worcester Polytechnic Institute

Leo Grande
Worcester Polytechnic Institute

Samuel John Pridotkas
Worcester Polytechnic Institute

Deep Learning for Data Privacy Classification

A Major Qualifying Project Report:
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

By:
Griffin Bishop
Leo Grande
Samuel Pridotkas
Harutyun Sadoyan

Advisors:
Randy Paffenroth
Stephan Sturm

Sponsor:
Bertram Dunskus, Aristo Consulting

Date: 4 November 2018

This project is submitted in partial fulfillment of the degree requirements of Worcester Polytechnic Institute. The views and opinions expressed herein are those of the authors and do not necessarily reflect the positions or opinions of Worcester Polytechnic Institute.

Abstract

The ubiquity of electronic services and communication has allowed organizations to collect increasingly large volumes of data on private citizens. This trend raises concerns about the privacy of individuals, and the transparency and availability of the data collected. To address these concerns, the European Union passed the General Data Protection Regulation, which gives EU citizens certain rights regarding their private information. Companies are required to ensure the protection and availability of this data to the individual. They must internally separate the personal data and protect it, which presents an arduous and time consuming challenge. This project explores a number of machine learning techniques to help automate this classification process into three distinct privacy tiers: Non-Personal information, Personal information, and Sensitive Personal information. After testing a number of models, we find that applying feed forward neural networks to bag-of-words representations of documents achieves the best performance while ensuring low training and prediction times. We achieve accuracy levels of approximately 89% on the dataset available, but hypothesize that better results could be achieved with a larger and more diverse corpus of documents.

Executive Summary

As society has become increasingly digital in its interactions, the issue of data privacy has become a growing public concern. The European Union seeks to address these concerns through implementation of the General Data Protection Regulation (GDPR), which requires companies to protect individuals' Personally Identifiable Information (PII). The GDPR defines three categories of data: Non-Personal information, Personal information, and Sensitive Personal information, each with different requirements for their protection. For many corporations, compliance represents a great challenge; documents containing PII may be distributed throughout large filesystems, and manual organization by the above categories could take hundreds or even thousands of man hours in large companies. An automated method for privacy classification of arbitrary text data is, therefore, an extremely beneficial instrument for such companies. In this report, we explore deep learning methods for this text classification task, and work with a corporation to develop a system for use in practice.

First we must convert documents from many formats commonly used by businesses (PDF, Word, PowerPoint, Excel, etc.) into plain text. Since many documents consisted of images of text, our system uses optical character recognition when applicable to extract the relevant plaintext. Next, the system produces a bag-of-words representation of each document and calculates the term frequency-inverse document frequency (TF-IDF) of each word. Intuitively, this statistic is intended to reflect the importance of a given word in a document with respect to a corpus of documents [1]. At this point, each document is represented as a vector with tens of thousands of entries. With a larger data set, this number would be much higher, approaching the number of words in the subject language. The next step, Principal Component Analysis (PCA), seeks to reduce this dimensionality and reduce noise while preserving the important features. We find that most of the variance can be preserved while reducing the dimensionality to under 500.

After the dimensionality reduction step, the system uses a feed forward neural network (FFNN) with two hidden layers of 128 and 32 neurons, respectively, to predict the privacy classification for a given document. We explore a large range of different models for this classification task, but find that FFNNs have the best performance with far lower training and prediction times.

The system described above allows prediction of the correct privacy classification for a given document in our test set over 89% of the time. In practice, such a system could be used to evaluate the degree to which each subset of a file system contained Personal or Sensitive Personal information or assist with manual classification of the documents. To further help with the latter task we devise a confidence rating measurement by which we can grade each of the system's predictions. For each document, we not only produce a prediction of the privacy classification, but the degree to which our system is confident in its prediction as a percentage. With this, a corporation can decide on a comfortable confidence level, and leave documents below that confidence level to be classified manually. In the test set, we find a strong positive correlation between the model's confidence and accuracy. For the most confident documents, our system makes the correct prediction over 99% of the time.

In addition, predicting privacy classifications for documents as a whole, our team also explores methods for predicting which groups of lines within a given document contain Personal information. To accomplish this, we define a procedure for labelling lines within documents based on HTML. We combine this labelling scheme with the concept of n-lines, which allows contextual information to be included in training examples. Using this procedure, our system predicts whether a given group of 8 lines contains PII at over 79% accuracy in our test set.

Contents

List of Figures

List of Tables

Glossary

MSE is the Mean Squared Error
stddev is the standard deviation

Notation Dictionary

Upper case variables such as X are random variables
Vectors are bold, whereas scalars are not
Hat variables such as $\hat{y}$ are predictions

$\mathbb{R}^n$ is the set of real-valued n-dimensional vectors
$\mathbb{R}^{n \times m}$ is the set of real-valued n by m dimensional matrices
$\mathbf{x}$ is the input vector such that $\mathbf{x} = [x_1, x_2..., x_n]^T$ and $x \in \mathbb{R}^n$
y is the ground-truth output label
$\hat{y}$ is the predicted output label
ϵ is the irreducible error
$\hat{f}$ is the approximated model mapping $\mathbf{x}$ to an output
$E(X)$ is the expected value of the random variable X
P(A) is the probability of an event A
P(A|B) is the probability of an event A given B
L is a loss function between two vector arguments of the same dimension
α is the learning rate
I_G is the Gini Impurity measure
$\mathbf{w}$ is a matrix of weights
r is the recall
p is the precision
σ is the activation function
c is the confidence in a prediction

1 Introduction

More than half the population of the world uses the Internet, generating massive amounts of data floating between endpoints. This data is often stored, analyzed, and utilized by Internet based corporations to meet a specific need. At times, the scope and pervasiveness of this data collection raises ethical privacy concerns, which led to the European Union (EU) releasing the General Data Protection Regulation (GDPR) in May of 2018 to protect the data and information of private citizens. The regulation aims to give citizens more control over their data, and add extra protections for Personally Identifiable Information (PII). Data curated in the EU falls under three categories: Non-Personal, Personal, and Sensitive Personal. Any data that can lead to the identification of one's identity is considered to be PII, yet this data can then be categorized further as either being in the Personal or Sensitive Personal class. Hence, any data that does not identify an individual falls under the Non-Personal class. These rules apply to any corporations that process or store data of EU citizens and carry the possibility of lawsuits or large fines if violated.

The GDPR presents a technical challenge for affected companies due to the fact that most information being held was not previously labeled as being in one of the three respective classes. Companies with larger databases can potentially have thousands, if not millions, of documents that have to be analyzed and labeled to comply with these regulations. This process is heavily tedious and requires meticulous inspection so to not miss any PII that could be present. The extensiveness of this process in turn costs companies money due to the allocation of resources to comply with these rules. The goal of our project is to help automate the document classification process through machine learning, which in turn will greatly help all entities trying to update their databases.

The system we designed is a machine-learning model trained to recognize and categorize PII using Natural Language Processing. The first step of this process was the classification of documents from a dataset that was given to our team mainly consisting of resumes, job postings, and company announcements. Through the determination of document classification, our system then had a set of data that it could learn from. The model is trained to extract relevant features from the text contained in documents. The documents, which come in many formats, are first converted to text, then undergo preprocessing to extract high value information, and are finally fed into the model to be categorized. We experimented with a number of different models and preprocessing techniques to find the optimal combination.

2 Background

This section first describes the legislation that motivates the project. Additionally, we explain the theory behind the machine learning and preprocessing techniques and models we use in Sections 3 and 4. Finally, this section covers the metrics we implement to evaluate model accuracy.

2.1 General Data Protection Regulation

The General Data Protection Regulation (GDPR) was approved by the European Union Parliament on April 14, 2016 in an attempt to regulate and standardize data protection laws. This regulation was put into full effect on May 25, 2018, and sets guidelines on how data is stored and shared for EU citizens [2].

The GDPR regulates the privacy of an individual's data held by an entity, such as an individual, organization, or business outside of a Personal setting. For example, if an individual's resume is stored within a company's database, it is subject to the GDPR. However, if an individual holds a Personal conversation over private email with a friend, then this does not fall under the GDPR [3].

The installation of these new rules focused on data privacy had a profound effect on business in the European Union. Numerous documents were released after the official passing of the GDPR to help get companies ready for the May 25, 2018 enforcement deadline. The regulation requires companies to shield certain types of data and not distribute it without explicit permission. Furthermore, citizens of the EU are allowed to query companies about what information these organizations have on them and can even request to delete said information. Penalties for GDPR infractions include a warning, a seize of data processing operations, or a fine depending on the severity. Data must be deleted if:

- The data collected were when the individual was a minor (under 18 years of age).
- The data were collected illegally.

However, companies do not have to delete this information if:

- Deleting the information discredited the company's freedom of expression.
- The data must be kept due to legal obligations.
- Deleting the information would discredit public interest (public health, history, statistical, etc. research data).

The request to remove data can be changed or denied based on the technical measures the company uses to protect said data. This can occur if the data are sufficiently anonymized to satisfy the individual, or is no longer identifying.

The GDPR divides data into three different categories:

- Non-Personal Data
  - Any data that does not directly or indirectly identify an individual.
- Personal Data
  - Any information that relates to an identified or identifiable person, directly or indirectly. This extends to concrete information regarding identification numbers along with physical, physiological, mental, economic, cultural, or social identity. I.e. if the data can be used to identify an individual based off of the categories above, it is Personal.
- Sensitive Personal Data
  - Data stored on an individual that identifies said person's racial or ethnic origin, political opinions, beliefs based on religion and/or philosophy, membership of a trade union, and data regarding one's sex-life and health.

These categories are tiered in ascending order of Non-Personal Data, Personal Data and Sensitive Personal Data. If both sensitive Personal and Personal data appear in a document then the entire document is considered to be a Sensitive Personal Document [4].

2.2 Machine Learning Overview

Machine learning is a field within computer science focusing on algorithms that improve themselves without being explicitly programmed. When designing a model, we begin with the assumption of some true universal relationship $Y = f(\mathbf{X}) + \varepsilon$ between the input vector $\mathbf{X}$ and the output label Y. The constant, ε, represents the irreducible error latent in any model. The task of a machine learning system is to approximate this relationship with its own function, $\hat{Y} = \hat{f}(\mathbf{X})$, where $\hat{Y}$ is the predicted output and $\hat{f}$ is our model. Most models come in one of two variants: regression or classification. Regression is the prediction of continuous values, such as height or weight. Classification, on the other hand, is the prediction of discrete categories, such as dog breed or color. This project focuses on the latter.

2.2.1 The Bias-Variance Trade-off and Cross Validation

The total error of a model is the sum of error due to *bias*, error due to *variance*, and the *irreducible error* ε. As mentioned above, we know that Y is the true equation and $\hat{Y}$ is our approximation. The error is then equal to the difference between them (squared to account for signs and give a more useful measure): $E[(Y - \hat{Y})^2]$. The derivation is below: [5]

$$\begin{aligned} E[(Y - \hat{Y})^2] &= E[(Y - \hat{f})^2] \\ &= E[Y^2 + \hat{f}^2 - 2Y\hat{f}] \\ &= E[Y^2] + E[\hat{f}^2] - 2E[Y\hat{f}] \end{aligned} \tag{1}$$

Now we can expand this into:

$$\begin{aligned} E[Y - \hat{Y}^2] &= E[(Y - E[Y])^2] + E[Y]^2 + E[(\hat{f} - E[\hat{f}])^2] + E[\hat{f}]^2 - 2E[Y\hat{f}] \\ &= E[(Y - E[Y])^2] + E[(\hat{f} - E[\hat{f}])^2] + E[\hat{f}]^2 - 2E[Y\hat{f}] + E[Y]^2 \end{aligned} \tag{2}$$

By assumption, since f is fixed, $E[f] = f$ and $E[\varepsilon] = 0$:

$$\begin{aligned} Y &= f + \varepsilon \\ E[Y] &= E[f + \varepsilon] \\ E[Y] &= E[f] + E[\varepsilon] \\ E[Y] &= f + 0 \end{aligned} \tag{3}$$

Also since f is fixed and $\hat{f}$ is independent of ε:

$$\begin{aligned} 2E[Y\hat{f}] &= 2E[(f + \varepsilon)\hat{f}] \\ &= 2(E[f\hat{f} + E[\varepsilon\hat{f}]) \\ &= 2(E[f\hat{f}] + E[\varepsilon]E[\hat{f}]) \\ &= 2(E[f\hat{f} + 0E[\hat{f}]) \\ &= 2fE[\hat{f}] \end{aligned} \tag{4}$$

So:

$$\begin{aligned} E[\hat{f}]^2 - 2E[Y\hat{f}] + E[Y]^2 &= E[\hat{f}]^2 - 2fE[\hat{f}] + f^2 \\ &= (E[\hat{f} - f])^2 \end{aligned} \tag{5}$$

Finally:

$$\begin{aligned} E[(Y - \hat{Y})^2] &= E[\varepsilon^2] + E[Y]^2 + E[(\hat{f} - E[\hat{f}])^2] + E[\hat{f}]^2 - 2E[Y\hat{f}] \\ &= E[\varepsilon^2] + E[(\hat{f} - E[\hat{f}])^2] + E[\hat{f}]^2 - 2E[Y\hat{f}] + E[Y]^2 \\ &= E[\varepsilon^2] + E[(\hat{f} - E[\hat{f}])^2] + (E[\hat{f}] - f)^2 \end{aligned} \tag{6}$$

This error is split into three major parts. The irreducible error is random noise with an expectation of zero. $(E[\hat{f}] - f)^2$ is the *bias*. This is the degree to which the model failed to capture the underlying relationship between the predictors and the output. $E[(\hat{f} - E[\hat{f}])^2]$ is the *variance*. This is a measure of how sensitive the model is to noise within the dataset. As shown on Figure 1, as the complexity of the model increases, bias goes down while variance goes up.

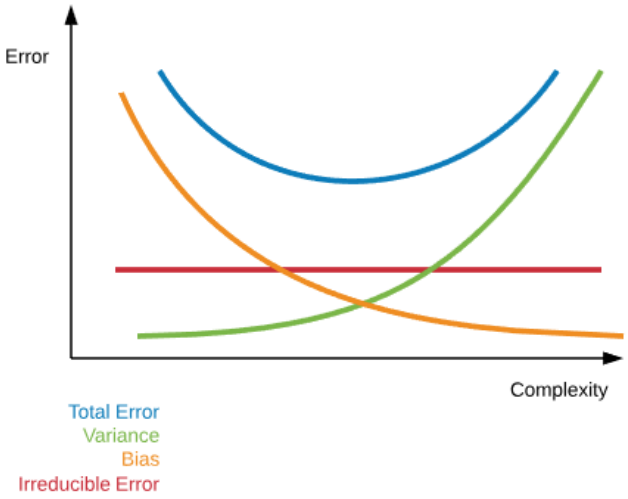


Figure 1: Graph of Bias-Variance Tradeoff

When total error is at a minimum when bias and variance are balanced. If the algorithm is too complex and the variance is high, it will learn to model the specific sample of data very well, while failing to approximate the true universal function. This is called *overfitting*. Figure 2 shows an example of this.

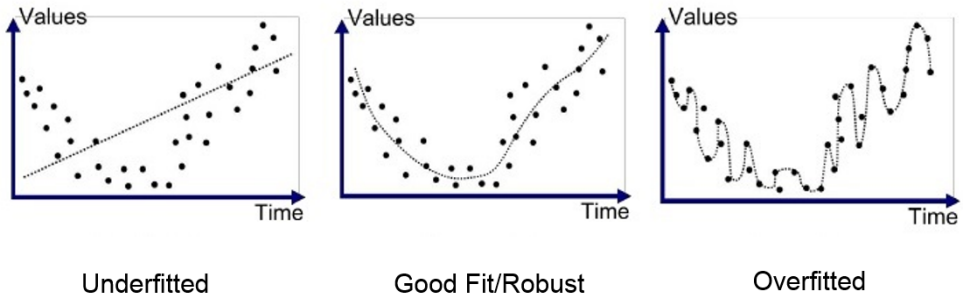


Figure 2: An example of an underfit model on the left, balanced model in the middle, and overfit model on the right. Graph taken from [6]

The graph on the left of Figure 2 has high bias and is underfit. The linear equation used is unable to fully capture the complexity of the model, leading to a high total error. The graph on the right has high variance, and is overfit. The model perfectly captures only this specific sample of data, and would not generalize to new data. However, if we were to rank the three models by their error, the one on the right would seem like the best one, giving us an inaccurate and inflated measure.

To get a useful measure of the accuracy of our model we need to perform *cross validation*. Cross validation is the practice of testing the model using labeled data that the model has not yet seen. This wards against inflated accuracies, because the testing error will be sensitive to overfitting while the training error will not. In the rightmost model in Figure 2, the training error would be very low, but the testing error would be high, alerting us that the model overfit. We would see the best testing error with the model in the middle, despite that model having a higher training error. The relationship between testing/training error and the bias-variance tradeoff is further illustrated in Figure 3.

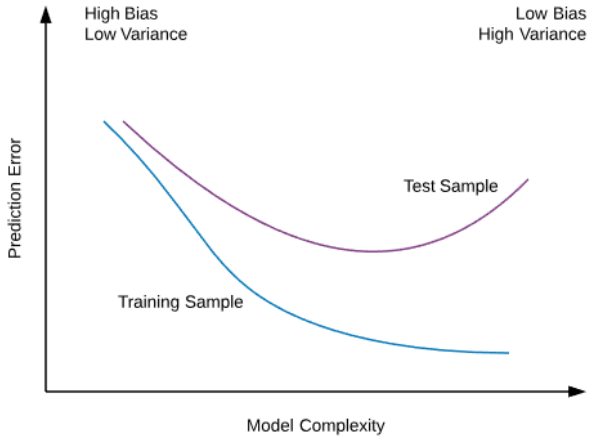


Figure 3: Test vs. Training Data

2.3 Preprocessing Techniques

Preprocessing involves transforming the data into a more meaningful representation to help filter out noise and aid in the knowledge extraction. Types of preprocessing include data cleaning, feature extraction, and dimensionality reduction. For this project we focus on techniques that have been historically successful for text classification: Word Count Vectorization, Term Frequency Inverse Document Frequency, Word2Vec, Principal Component Analysis, and Autoencoders.

2.3.1 Word Count Vectorization

Word Count Vectorization converts terms within a document to a vector of word counts. Word Count Vectorization is unique due to its simplified form of only accounting for each time a term appears. This high level vectorization is sometimes optimized by researchers to remove words that have greater than, less than, or equal to a certain word count, or to even group words of a certain count together to be utilized within a machine learning model [7].

2.3.2 Term Frequency Inverse Document Frequency

Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical weight used to evaluate the importance of a word in a collection of documents. This weight is calculated as the amount of times a term occurs in a document and how many documents that term occurs in

$$TF - IDF = \frac{d}{t} \log\left(\frac{o}{q}\right)$$

Where

- d is the total number of times a word appears in a document
- t is the total number of words in the document
- o is the total number of documents
- q is the total number of documents containing a specific word

First, the frequency of a word within the document is calculated, normalizing for document length. As seen in the formula above, the number of times a term appears is divided by the total number of terms found. The second half of the formula, inverse document frequency, measures how important a word is. All terms within the document are equal when evaluating term frequency. Yet, when it comes to highly common words, such as prepositions, inverse document frequency will scale the weights for these terms down. The words found scarcely throughout the document will in turn have a heavier weight. Once the TF-IDF preprocessing technique is finished, each document will be converted to word frequency vectors with decimal values between zero and one. A value closer to one means that the word occurs more frequently within the document and rarely within other documents, and has more importance. TF-IDF is widely used in text classification due to the power of this weighting system [1].

2.3.3 N-Gram Decomposition

N-gram decomposition is a preprocessing method that involves the combination of a sequence of n items. For example, when performing this decomposition on a given sentence, 'The cat jumps high.', where $n = 2$ the output would be [[The,cat], [cat,jumps], [jumps,high]]. This technique allows information about relationships between sequential items to be preserved, which can increase the accuracy of a machine learning model in certain cases. N-Grams are especially useful in text classification, because oftentimes words in sequential order hold meaning in their relationships to each other.

2.3.4 Word2Vec

Word2Vec is another word embedding preprocessing technique that was introduced by Google Inc. in 2013 [8]. Word2Vec requires a large text corpus and can be used to make a weighted guess for each word based on its surrounding terms. The model builds a vocabulary containing each word and their respective vector representation [9].

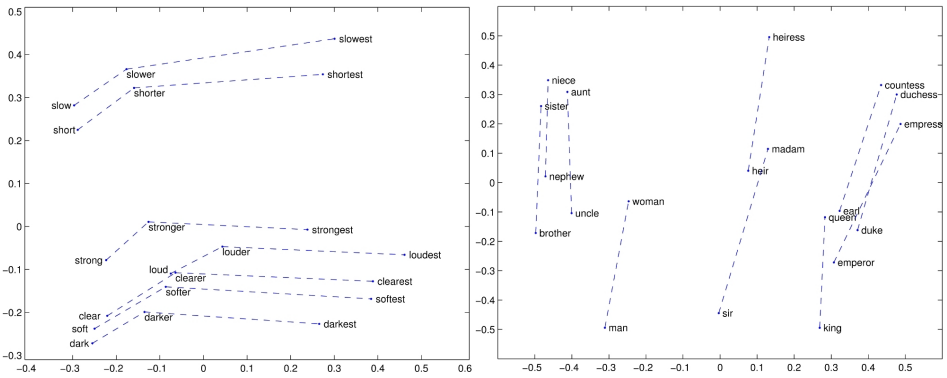


Figure 4: Graphical representation of the words within a Word2Vec created dictionary in a 2-Dimensional space. Figure borrowed from [9].

The distances between words in Figure 4 represent the relationship between them. For example, it becomes possible to mathematically assert that *king - man + woman* $\approx$ *queen*.

The vector representation of a word within the vocabulary shows a word's context in regard to each other word as number no higher than one. When two words commonly appear together, this number is closer to one. For example, if the word "shampoo" occurs next to "shower" many times in the text corpus, the vector representation when comparing these two words will be close to one. Word2Vec works well with text classification because it maintains the relationship between terms. A machine learning model can use this to its advantage by better understanding the meaning of words and correlating this to the classification of the full text [10].

2.3.5 Principle Component Analysis

Principal Component Analysis (PCA) is a statistical analysis technique often used for dimensionality reduction. The purpose of PCA is to map the data to a lower dimensional space, while maintaining the most variance. Figure 5 shows a dataset that was transformed along two Principal Components along the red and green axes. The red component explains the most variance, and green the second most. Now we can discard the green component to reduce to one dimension while losing the least amount of information.

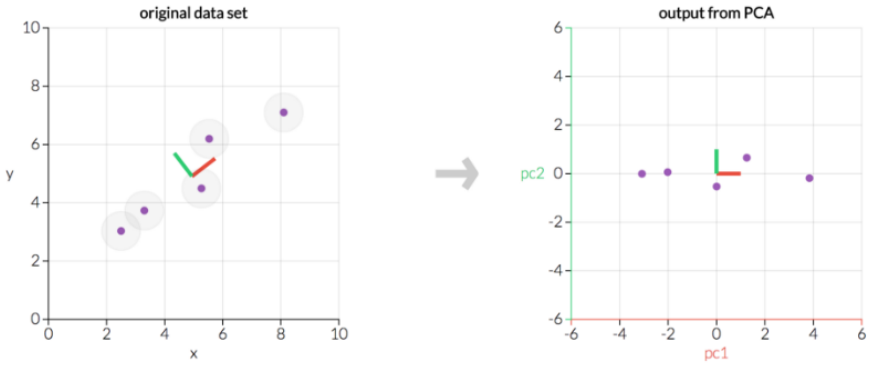


Figure 5: PCA dimensionality reduction example. This figure shows two principal components (Red and Green) along with the features (dots) in the graphs. The left graph is the original dataset, and the right graph is the output after performing PCA. Figure taken from [11].

Each of the Principal Components is a linear combination of the existing parameters of the form

$$v_1 = a_{11}c_1 + a_{12}c_2 + a_{13}c_3 ... a_{1n}c_n$$

Where:

- v_i is the i^{th} principal component
- c_n is the n^{th} original parameter
- a_n is the weight of that parameter in the component

The weights are calculated with the constraint that the sum of their squares must equal to 1. [12] PCA can be calculated through *Singular Value Decomposition*. SVD is a factorization of a real or complex matrix. Suppose there is an original matrix $m \in \mathbb{R}^{m \times n}$. Then there exist matrices such that $\mathbf{m} = \mathbf{u}\boldsymbol{\psi}\mathbf{v}^T$ [5] Where:

- $\mathbf{u} \in \mathbb{R}^{m \times m}$ is unitary matrix of the linear combinations
- $\boldsymbol{\psi} \in \mathbb{R}^{m \times n}$ is diagonal matrix of the singular values
- $\mathbf{v} \in \mathbb{R}^{n \times n}$ is unitary matrix of mappings back to high dimensional space

A matrix $\mathbf{A}$ is considered unitary if its conjugate transpose $\mathbf{A}^*$ is equal to its inverse, $\mathbf{A}^{-1}$ [13]. The values in the diagonal of $\boldsymbol{\psi}$ are sorted in descending order by the variance explained, and the lower values get discarded. Given this, $PCA(\mathbf{m}) = \mathbf{u}\hat{\boldsymbol{\psi}}$ where $\hat{\boldsymbol{\psi}}$ is $\boldsymbol{\psi}$ with the lower values set to 0. Each principle component is perpendicular to all of the previous ones. This allows us to reduce the original matrix to a new j dimensional space where j is the number of components we decide to keep.

2.3.6 Autoencoders

An autoencoder is a type of unsupervised machine learning method used for dimensionality reduction. Autoencoders work by learning a low-dimensional mapping $encode(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$ and a mapping back to the higher dimensional space $decode(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}^n$ (where $m < n$) such that the following quantity is minimized:

$$\sum_{i=1}^{j} L(decode(encode(\mathbf{x}_i)), \mathbf{x}_i)$$

- L is some loss function between two vector arguments in $\mathbb{R}^n$ (MSE for example)
- $encode(\mathbf{x})$ is a function $\mathbb{R}^n \to \mathbb{R}^m$
- $decode(\mathbf{x})$ is a function $\mathbb{R}^m \to \mathbb{R}^n$

- $\mathbf{x}$ is the input vector such that $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2..., \mathbf{x}_j]^T$ and each entry $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_j$ in $\mathbf{x}$ is $\in \mathbb{R}^n$
- n is the dimensionality of each training example $\mathbf{x}_i$
- m is the desired number of dimensions in the low-dimensional space ($m < n$)

In this way, autoencoders learn to compress the input data into a lower dimensional representation and then uncompress this to match the original data as closely as possible.

A single layer linear autoencoder should be identical to PCA, but autoencoders are not restricted to being linear. When using a non-linear activation function in the neurons, autoencoders can be expected to outperform linear dimensionality reduction methods on non-linear data. Figure 6 illustrates this with a comparison between PCA (a linear dimensionality reduction method) and autoencoders on the MNIST dataset. MNIST is a dataset of 60,000 images of handwritten digits where each image is 28 by 28 pixels. In the figure, the classes of the 10 handwritten digits appear as different colors. As shown, the autoencoder more clearly clusters and separates the classes than PCA [14].

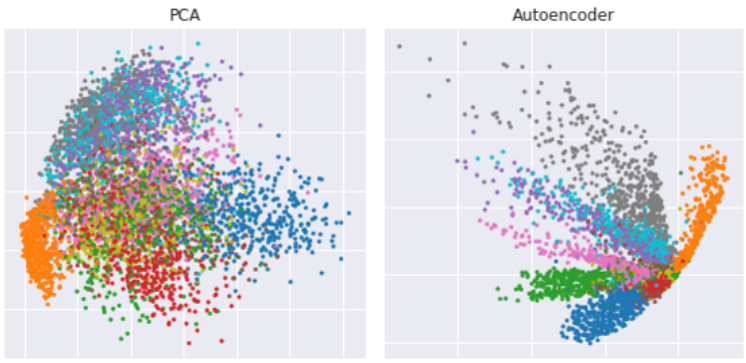


Figure 6: A comparison of the output of PCA versus autoencoder performed on the MNIST dataset

2.4 Models

Once the training data has fully undergone preprocessing it is used to train algorithms for classification. The models explored for this project include Naive Bayes, Random Forest, linear and nonlinear support vector machines (SVMs), Extreme Gradient Boosting, and Neural Networks.

2.4.1 Neural Networks

Artificial neural networks constitute a family of machine learning models based on automatically learning data representations. Methods like decision trees learn task specific rules which are used to split the input data into sub-categories at each node. In contrast, neural networks are characterized by not learning any task specific rules. This makes neural networks, while effective, difficult to interpret. The major downside to using neural networks is this lack of explainability.

Neural networks consist of an interconnected set of nodes organized into layers, with an input layer whose nodes take on the values of a given input vector, and an output layer which reveals the final computed values. In feed-forward neural networks, considered to be the simplest form of neural network, each node on a given layer is connected to every node on the next layer, and these edges do not form cycles. In this way, the input to any given neuron is a linear combination of the outputs of the neurons in predecessor layer with a set of trained weights. Next, an *activation function* is applied to the result of this linear combination, which defines the output of a node given an input.

The output $\mathbf{z}_l$ of a given layer l can be computed as follows:

$$\mathbf{z}_l = \sigma\Big(\sum_{i=1}^{n} \mathbf{w}_i^l \mathbf{h}_i^l + \mathbf{b}_i^l\Big) \tag{7}$$

where

- $\mathbf{w}_i^l$ is the i^{th} weight of the l^{th} layer

- $\mathbf{b}_i^l$ is the bias weight of the l^{th} layer
- $\mathbf{h}_i^l$ is the i^{th} output of the $l-1^{th}$ layer
- n is the number of nodes in the previous layer
- σ is the activation function

This process may be repeated on successive layers to compute the final output of the network.

By using a non-linear activation function, neural networks are able to learn to solve non-trivial (non-linear) problems through *backpropagation*. Backpropagation involves calculation of the gradient of the loss function with respect to the weights in the network in order to iteratively update each of them. Backpropagation is named as such because in networks with multiple layers between the input and the output (deep neural networks) the total error is calculated from the output and then "propagated" backwards through the predecessor layers. A useful way to intuit this is to think about the effect of a single neuron. If we change the i^{th} neuron in layer l by a small amount Δz_i^l, the output of the loss function will change by an amount $\frac{\delta f_{loss}}{\delta z_i^l}\Delta z_i^l$. If the goal is to minimize the resultant loss, one can simply adjust the weight of the neuron by a constant multiple of $\frac{\delta f_{loss}}{\delta z_i^l}\Delta z_i^l$. This constant multiple is referred to as the *learning rate* of the network. If too small, the weights will update very slowly, and learning will take a long amount of time. If the learning rate is too high, however, minima of the loss function may be skipped or 'overshot', causing the network to fail to find a convergent solution. [15]

2.4.2 Activation Functions

In neural networks, an activation function defines the output of a node when given an input. Three classes of activation functions are the *Rectified Linear Unit* (ReLU), *Sigmoid* and *Softmax* functions.

ReLU activation function is defined as $R(x) = max(0, x)$ where x is the input to the node. This activation function is non-linear, *sparse* and very efficient computationally. ReLU is sparse because it does not activate for all negative values. This leads to models that are faster and are less likely to overfit. However during training, this can lead to nodes that always output the same value for any input. This is referred to as "dying" and when this happens, the dead node no longer contributes to the training of rest of the network. Since the gradient of ReLU is zero when $x \leq 0$, the node will not change the weights of the input and won't recover from this state [16].

The Sigmoid activation function is used in multi-class classification tasks for which the probability of the output classes is assumed to be independent. When displayed visually on a graph, the Sigmoid function exists between zero and one on the y-axis, and is shaped like an S as seen in Figure 7.

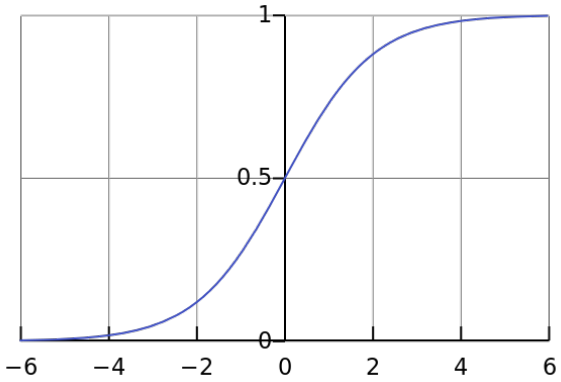


Figure 7: Graph visualizing the Sigmoid activation function. It is S shaped with range between zero and one. Taken from [17].

This graph is given by the following function:

$$A(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid is useful for multi-class classification because its output (for each class) is between zero and one, an output range which is necessary for interpretation as a probability. Furthermore, when the x-axis values change, especially between negative two and two, the y-value changes drastically. Therefore, any independent variable will trend toward either zero or one, which helps determine the classification outcome [17].

The softmax function is a generalization of the sigmoid function used as the last hidden layer in multi-class classification where the output probabilities are assumed to be dependent. The softmax function produces probabilities over a number of categories such that the probability sum to one [18]. Given an arbitrary real valued *K-dimensional* vector $\mathbf{z}$, the function outputs a K-dimensional vector $\boldsymbol{\sigma}(\mathbf{z})$ whose entries fall into the range $(0, 1)$. The softmax function $\boldsymbol{\sigma}(\mathbf{z})_j$ is computed as follows:

$$\boldsymbol{\sigma}(\mathbf{z})_j = \frac{e^{\mathbf{z}_j}}{\sum_{k=1}^{K} e^{\mathbf{z}_k}} \textit{for j=1,2,...,K}$$

where

- $\mathbf{z}$ is the real-valued vector input
- and K is the dimension of both the input and output vectors

2.4.3 Dropout

In general, neural networks with multiple layers are extremely powerful. By using a high number of layers, bias can be reduced arbitrarily. However, this makes the class of models especially prone to overfitting. *Dropout* is a technique commonly used to combat this while retaining model complexity. When using dropout, every iteration of backpropagation involves dropping each neuron (and its connections) with a certain probability, usually around 20%. As a result, the model trains as an average of multiple thin networks. This reduces the impact of noise and can prevent multiple neurons from learning to represent the same feature (called co-adaptation). [19]

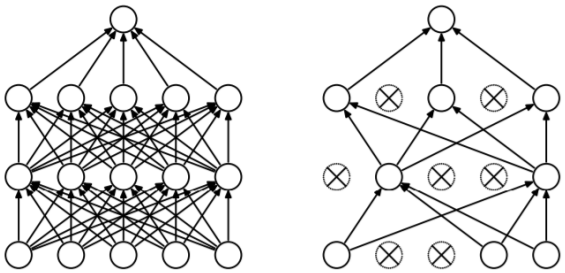


Figure 8: Left: Deep neural network. Right: a thin = network; the result of applying dropout to the network on the left. Crossed units are dropped. This figure was borrowed from [19].

2.4.4 Naive Bayes

Naive Bayes is a classification algorithm based off of Bayes' Theorem. This theorem calculates the probability of an event based off of information in samples using the following equation:

$$P(c|x) = \frac{P(x|c)\,P(c)}{P(x)}$$

In this equation, $P(x|c)$ is the conditional probability that the predictor, x, is a given class, c. $P(c)$ is the prior probability of the class, and $P(x)$ is the prior probability of the predictor. $P(c|x)$ is the posterior probability of the class, given a predictor. The posterior distribution is calculated based off prior observations, which is defined as:

$$P(c|x) = P(x_1|c)P(x_2|c)...P(x_n|c)P(c)$$

Using this, Naive Bayes calculates the probability that each example is a certain class and tags it as the one with the highest probability. This algorithm is computationally fast and performs well in multi-class classification [20].

2.4.5 Gradient Boosting

Boosting is a general term for algorithms that create a strong learner from a set of weak learners. In *gradient boosting*, each instance of a model is created sequentially through and makes a prediction for classifying a small subset of data.

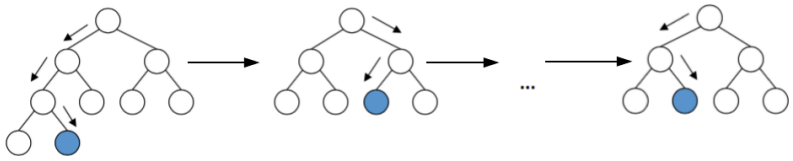


Figure 9: Visualization of how boosting sequentially creates models based off of their previous version. Figure taken from [21].

Within gradient boosting, at each iteration, the model outcomes are weighed based on the previous instance. Correct outcomes are weighted lower than incorrect outcomes, and from here the model will then focus more on correcting the higher weighted outcomes. The goal for each instance of a model is to minimize the Mean Squared Error loss function

$$Loss = MSE = \sum (\mathbf{z}_i - \mathbf{z}_i^p)^2$$

Where:

- $\mathbf{z}_i$ is the i^{th} target value
- $\mathbf{z}_i^p$ is the i^{th} prediction

This occurs by updating an instance of a model based on the learning rate that is specified in the algorithms parameters:

$$\begin{aligned} I_G &= \sum_{i=1}^{j} P(k) \sum_{k \neq i} P(k) \\ &= \sum_{i=1}^{j} P(i)(1 - P(i)) \\ &= \sum_{i=1}^{j} (P(i) - P(i)^2) \\ &= \sum_{i=1}^{j} P(i) - \sum_{i=1}^{j} P(i)^2 \\ &= 1 - \sum_{i=1}^{j} P(i)^2 \end{aligned} \tag{8}$$

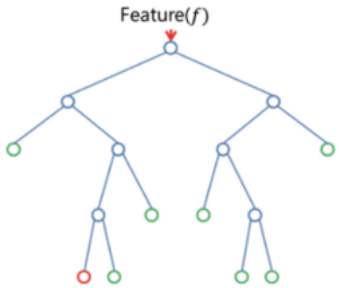


Figure 10: A visualization of a single decision tree that has a parent node and the top and multiple child nodes. This image is from [22].

The decision trees are created using *bagging*. Bagging (Bootstrap aggregating) is the process of drawing m random datasets, with replacement, from the original training set D. Each decision tree trains on a different set m[i] in parallel [23]. Each tree is independent of the others. After all of the decision trees are created, the forest makes a decision by averaging the predictions made by each tree.

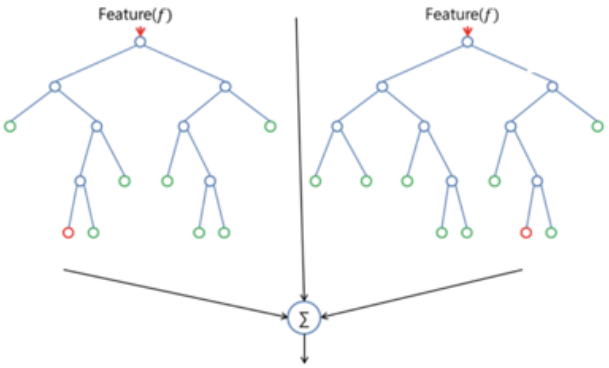


Figure 11: Decision Trees within a forest that are created in parallel and come together to average out a final accuracy measure. Image used from [22].

Random forest is different than a normal decision tree algorithm since instead of searching for the feature with the highest importance, it instead searches for the best feature within a random subset of features through the creation of multiple decision trees. The algorithm will create a wide range of rules to determine how to execute classification and then brings these predictions altogether in the end. Since the trees are created randomly, the algorithm is safe from overfitting (assuming tree depth is limited). Random forest encompasses and creates a wider range of rules across a dataset in an overall better manner than a regular decision tree algorithm [22].

2.4.6 Support Vector Machines

Support Vector Machines are a class of algorithms designed to find a *hyperplane* that separates sets of data into different classes. New data is then mapped and classified by the side of the hyperplane the data is located. A hyperplane is a subspace that is one dimension less than its parent space. For example, in a two dimensional space a hyperplane is a line. It is defined by the equation $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p = 0$ where x is a point on the hyperplane [23]. This is demonstrated in Figure 12.

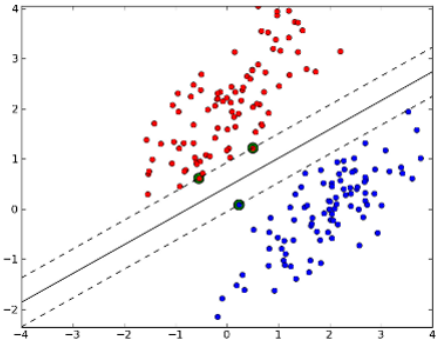


Figure 12: The Definition of Hyperplanes for SVMs. Figure used from [24].

SVMs find the hyperplane that has the largest distance, called the margin, between the hyperplane and the nearest data points, which are called support vectors. The larger the margin that is found, the more confidence new data can be classified with [25]. SVMs classify nonlinear data by using *kernel functions* which map the data to a higher dimension so that a hyperplane can be defined.

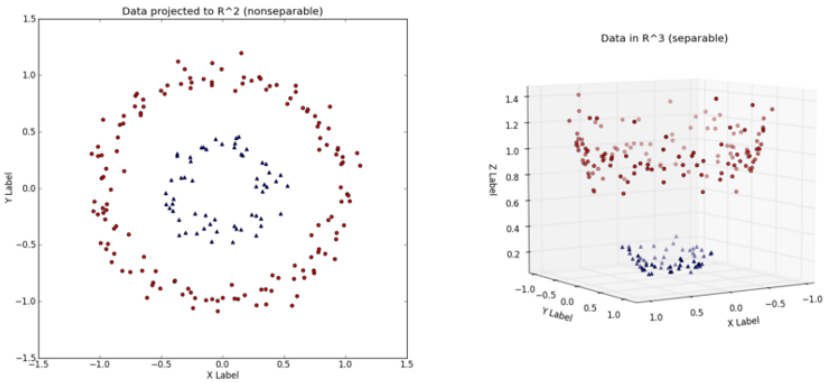


Figure 13: Example of the Kernel Method for Support Vector Machines. Borrowed from [24].

In Figure 13 the dataset cannot be separated linearly. However, when the data is mapped to three dimensions, a plane can now be used to separate the data. One class of kernels is the *Radial Basis Function* (RBF) kernel. RBF calculates the *Euclidean*, straight-line, distance between two points in higher dimensions [26] using the equation $\phi(x, t) = |x - t|$. Where t is the point, called the *center*, that the distance is being calculated from.

2.4.7 Mini-Batch Stochastic Gradient Descent

Autoencoders, Neural Networks, and Support Vector machines use Mini-batch Stochastic Gradient Descent. *Gradient Descent* optimizes a cost function by randomly weighting data in a training set and then calculating the *gradient* based off of these weights. The gradient is the derivative of the cost function. Based off of the gradient the data is re-weighted and the process is repeated until the gradient is equal to zero and the cost function is optimized [27]. The rate at which the weight of the data is changed is called the learning rate. Since gradient descent uses all data points in each iteration of calculating the gradient, this is computationally expensive. To avoid this, stochastic gradient descent uses one random sample to calculate the gradient in each iteration. However, this process is very noisy. A trade-off between the two is mini-batch stochastic gradient descent. This form uses n-samples in each iteration, where n is greater than one and less than or equal to one thousand.

2.5 Accuracy Evaluation

Evaluating how well a model performed classifying data is the final step of the machine learning process. Yet, unless the training data is perfectly balanced across all categories, the accuracy of a system can be misleading.

When the data categories are highly unbalanced, with 90% of type A and 10% of type B, a simple accuracy can be misleading. A model could trivially achieve an accuracy of 90% by simply classifying everything as type A. In a situation like this, *F scores* are a much better indicator of model performance. The F score is a measure of accuracy that considers *precision* p and *recall* r. Precision is the number of true positives, divided by the number of total positives. Recall is the number of true positives, divided by the number of all values that should have been positives.

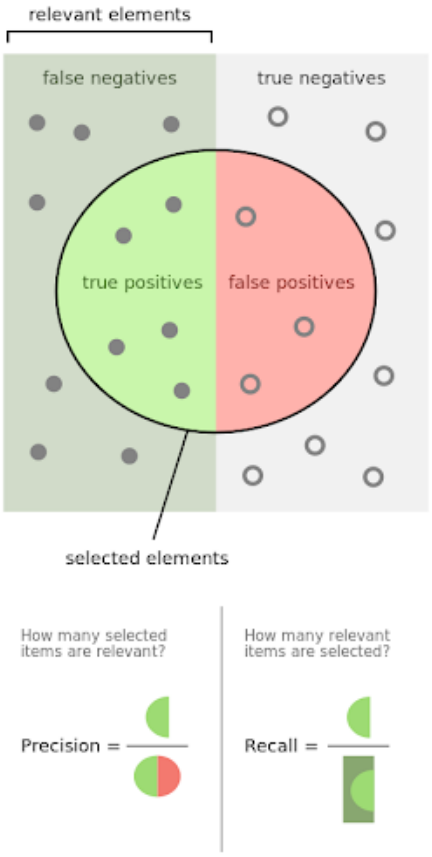


Figure 14: Chart showing the difference between true/false positives and negatives along with describing what precision and recall are when it comes to determining accuracy. This figure was taken from [28].

The general formula for an arbitrary β is:

$$F_\beta = (1 + \beta^2)\frac{pr}{(\beta^2 p) + r}$$

Where

- p is precision
- r is recall

Commonly used variants are the F_2 score, which emphasizes recall and false negatives, and the $F_{1/2}$ score, which emphasizes precision and false positives [23].

3 Methodology

3.1 Data Curation

Training a machine learning model requires a set of existing data that can be used to teach the system how to behave. For our project, we classify the overall documents into the GDPR defined categories and identify the different types of PII in each document. However, the data we use to train our model was not previously labeled with the information needed to complete these two goals.

The data available to our team consists of a wide range of file types. These files are rarely in a usable text format and mostly consist of PDFs, PowerPoints, and Microsoft Word files. To deal with the varied formats, an *Optical Character Recognition* (OCR) algorithm is used to convert every file into a text file.

Once the files are converted, manual classification is done on the document and then on the line-by-line levels. The documents are separated into Non-Personal, Personal, and Sensitive Personal categories by identifying the type of PII within the document as a whole. The line-by-line system, on the other hand, uses customized HTML tags that label PII found in each line. Below are the main categories of PII seen within the training documents. While there are other categories specified in the GDPR, they either do not exist in the training data or appear only a few times. These scarce categories are still labeled, but are not considered in our model.

1. Personal
   (a) <name>
   (b) <id-number>
      i. Health care id numbers, SSN, bank account numbers
   (c) <location>
      i. Current location, Personal address, and work address
   (d) <online-id>
      i. email and other edge cases
   (e) <dob>
   (f) <phone>
   (g) <professional>
      i. Profession, workplace, education
   (h) <physiological>
   (i) <social>
2. Sensitive Personal
   (a) <criminal>
   (b) <origin>
      i. Birthplace, ethnicity, nationality
   (c) <health>
      i. Disabilities, hospital visits, health insurance claims
   (d) <religion>

Labeling with HTML tags allows for a standardized format, high granularity, and ease of use. For example, if there is a 500 line document that only has 10 lines of PII, the model will know to recognize those specific lines. Furthermore, the labeling system encompasses multiple lines with minimum labeling. When the HTML tags are parsed, each line encompassed within a tag is identified as having that specific category. The parser can also handle lines with multiple categories, which we encounter often in the dataset.

There are some downsides to the Line by Line labeling. Our team of four has to go through each individual document, ranging on average from 50-600 lines, which is a very time consuming process. The problem is further aggravated by the language barrier since more than 95% of the documents in the training set are in German.

Below is an example of a labeled line.

<name>My name is <data>John Smith</data></name>

Please see Appendix A for further examples.

3.2 Model Selection

Next we select several machine learning algorithms to explore, focusing on ones that have performed well on text-classification tasks in the past. For each of these models, we also use several different preprocessing techniques and optimize their *hyperparameters* using *grid search* or *random search*. Hyperparameters are the properties that affect the training of the algorithms including learning rate, loss function, or number of hidden neurons. Grid search runs each model using every combination of a given range of hyperparameters to determine which ones yield the best results. Random search samples a number of combinations of hyperparameters to determine which is the best. For example, if there are 10000 different combinations it will randomly select 1000 combinations. Random search is less computationally expensive and in practice, has been found to produce comparable results to grid search [29].

3.3 Model Evaluation

Based on the GDPR, it is more detrimental for a Personal document to be classified as Non-Personal than the reverse. Therefore, we want our models to err on the side of caution. In order to represent this distinction in our model evaluation, we use F scores with a beta of 2 (F_2 score). The higher beta value emphasizes the recall, meaning that the detriment of false negatives contributes more than the benefit of false positives into the final score.

For measuring the accuracy of our models, we use *K-fold cross validation* with 5 folds. This allows us to minimize the variability in our results and receive a more accurate depiction of the true performance of each model. Within these folds, or sections, we reserve one for testing and the other four for training. We alternate through each of the five combinations to produce five measures of accuracy and average them to produce one final result. Without the implementation of K-fold cross validation, accuracy measures oscillate as much as five to ten percent between runs within our system. K-fold cross validation drastically reduces this range and produces more representative accuracies.

3.4 Confidence Metric

The major objective of the project is to create a model that can reliably classify documents into the three categories as described above. The intended usage of such a classifier would be to eliminate or reduce the human effort required to manually organize a corporation's data for GDPR compliance. While our models achieve accuracies of close to 90%, there is still a degree of error possible. If an organization must be more sure that a given file contains or does not contain Personal or sensitive Personal information, they may have to manually analyze the file. To streamline this process and provide more insight into the output of our classifier, we calculate a custom confidence measure for each prediction. Our neural network model is useful for this because it outputs the softmax probability over the three different privacy categories. The softmax function is described in Section 2.4.2.

We theorize that if a single category probability is high and the others were low, this represents high confidence in the prediction. Conversely, if more than one category probability is high, or the difference between all three is low, then this represents a low confidence for the prediction. We find that taking the standard deviation of the entries in the normalized vector results in a very workable measure for confidence. The confidence in a prediction is defined as a function $c(\mathbf{s})$ where $\mathbf{s}$ is the real-valued 3-dimensional vector output of the feed forward neural network representing a probability for each privacy category. This calculation is shown below:

$$c(\mathbf{s}) = stdev\left(\frac{\mathbf{s}}{|\mathbf{s}|}\right)$$

Figure 15 is a graph demonstrating the high correlation between our confidence measure (orange) and the observed accuracy for documents equal to or above that accuracy (blue). The two plots are normalized to the same range (0 to 1) in order to see this correlation.

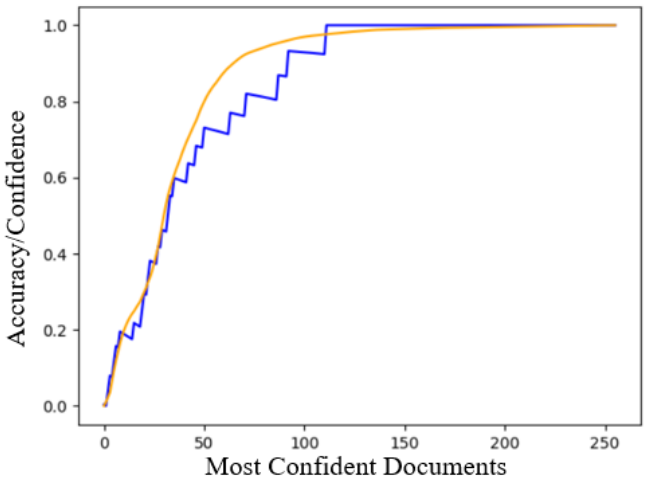


Figure 15: Correlation of the confidence metric(orange) with the actual accuracy(blue)

Using this metric, one can decide an arbitrary cut off point for document confidence. In the example from Figure 15, the set of 150 most confident documents have an accuracy of over 98% (F_2 scores of 1, 0.970, 0.985 for Non-Personal, Personal, and sensitive Personal categories, respectively). In practice, this makes it possible to select low confidence documents that might require human analysis. Additionally, this confidence metric is useful for the second part of the labelling process. Before doing the bulk of the work on our models, we label a large number of documents using the procedure described above. To do this, we randomly pick documents from a collection of unlabeled documents in no order. After training models and realizing it is beneficial to have a larger labeled corpus, we use the confidence metric to decide which documents to focus on. That is, we organize the documents in reverse confidence order, and focus on the documents our model is least confident in. By doing this, we avoid giving the model more training examples of the type it can already accurately predict. Instead, we provide more training examples of the type the model struggles with.

3.5 N-lines

To improve the accuracy of our model we implement the concept of *n-lines*, or n-grams at the line level. N-lines is a range of lines that creates overlapping continuous sequences, a rolling window, of the lines in the document. For example, with the dataset below:

Line1
Line2
Line3
Line4
Line5

and an n of 2 we would get: [Line1 Line2], [Line2 Line3], [Line3 Line4], [Line4 Line5].

However, a range of 2-3 would give us: [Line1 Line2], [Line2 Line3], [Line3 Line4], [Line4 Line5], [Line1 Line2 Line3], [Line2 Line3 Line4], [Line3 Line4 Line5]

Most of our models perform better with n-lines as the extra information provides more context and more successfully captures multi-line data points, like addresses.

4 Results

After manually labeling data for three weeks, as described in Section 3.1, we accumulate 930 files worth of labeled data. Table 1 provides a complete breakdown of this data.

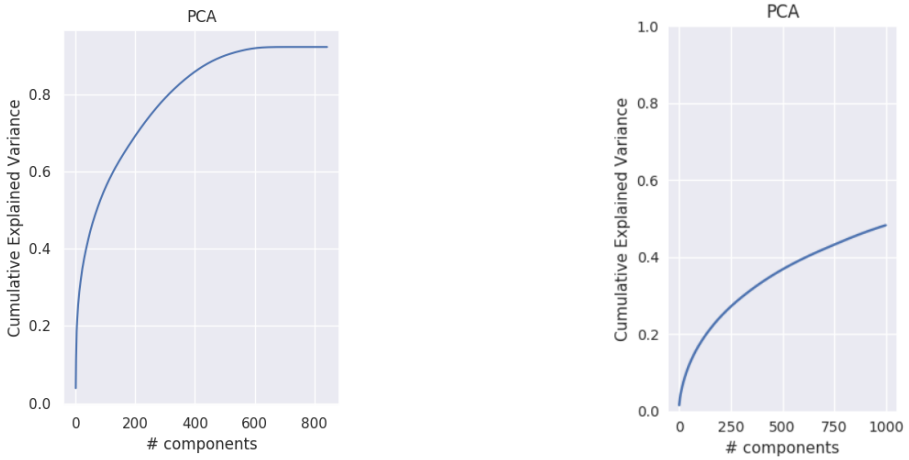
Table 1: Count of Labeled Data by Category

Total Lines	191219
Total Lines with PII	24292
Type of PII	*Number of Lines*
Professional	16522
Name	3698
Location	1972
Phone	678
Online-Id	544
Date of Birth	270
Id-Number	246
Origin	205
Health	133
Criminal	83
Physiological	30
Religion	12
Social	9

We use this data to train and test the models. This section describes the results of the models and the preprocessing techniques we use with them.

4.1 Preprocessing

Of the text processing and feature extraction techniques we use, some of the most effective methods include Word Count Vectorization, TF-IDF, and PCA. Word Count Vectorization and TF-IDF work well with our data to get a useful mathematical vector representation. PCA is able to reduce the dimensionality of our data while maintaining a high degree of the variance, helping reduce complexity and filter out noise. Most of our models then use this reduced data to increase computational performance and accuracy.



(a) PCA Document Level

(b) PCA Line Level

Figure 16: Explained Cumulative Variance vs. Number of Components

While some preprocessing methods have a positive impact on our model performance, others have the opposite effect. The first of these forms include the fixing of misspelled words. We manually find words within documents that are commonly misspelled due to either human error or

our OCR and correct these. We think that this is ineffective because we do not fix every misspelling within each document and do not have an expansive list of words that are possibly misspelled. Secondly, Word2Vec is used in the attempt to preserve the relationships between words. We create a dictionary of words based on the text corpus within our dataset, which in turn notes similarities between each and every word. Due to the text corpus being in multiple languages, we attempt to create our own language with the creation of this dictionary. Yet, Word2Vec is oftentimes ineffective unless the corpus is extremely large when creating a dictionary. If our text corpus exponentially increases, then Word2Vec may be effective. Word2Vec is the one preprocessing form we find that has a major negative impact. When using TF-IDF instead of Word2Vec, each model has at least an F_2 score that was .4 points higher than the latter. Each of these preprocessing techniques that fail are combined to try and increase their impact, yet again this is to no prevail.

4.2 Naive Bayes

Naive Bayes is a standard machine learning algorithm that is computationally fast and handles multi-class classification exceptionally well. This project's research revolves around the classification of documents and segments of documents into various PII categories, therefore relating to one of the strong points in Naive Bayes. The preprocessing techniques that are the most effective with this algorithm are a combination of Word Count Vectorization, and TF-IDF. The results of Naive Bayes are seen in Table 2.

Table 2: Naive Bayes Line by Line Results

	Non Personal	*Personal*	*Sensitive*
F_2 **Scores**	0.964	0.214	0

		Predictions		
	Confusion Matrix	*Non Personal*	*Personal*	*Sensitive*
	$Non - Personal$	26752	55	0
Actual Value	*Personal*	4692	1024	0
	Sensitive	99	3	0

As evidenced by the confusion matrix, Naive Bayes did not handle the unbalanced nature of the data. The predictions were heavily biased toward Non-Personal, with none of the Sensitives and less than a fifth of the Personals being classified correctly.

4.3 Random Forest

Random forest is a model that is known for being simple, effective, and resistant to overfitting. Furthermore, the preprocessing techniques we use with this model are Word Count Vectorization, TF-IDF, and PCA. Using Random Search for the hyperparameter tuning, the model works best with around 200 PCA components, and n-lines of two.

Table 3: Random Forest Line by Line Results

	Non Personal	*Personal*	*Sensitive*
F_2 **Scores**	0.878	0.796	0.662

		Predictions		
	Confusion Matrix	*Non Personal*	*Personal*	*Sensitive*
	Non Personal	7169	59	5
Actual Value	*Personal*	746	438	3
	Sensitive	3	1	137

4.4 Support Vector Machines

We test Linear and nonlinear SVMs because of their history of use in text classification tasks. Linear SVMs work the best with Word Count Vectorization and TF-IDF. We also use Stochastic Gradient Descent to optimize the model. Hyperparameter tuning indicates that linear SVMs work best with n-gram of 1-2 and n-lines of 5,6. The average accuracy is shown in Table 4.

Table 4: Linear SVM Line by Line Results

	Non Personal	*Personal*	*Sensitive*
F_2 **Scores**	0.937	0.816	0.549

		Predictions		
	Confusion Matrix	*Non Personal*	*Personal*	*Sensitive*
	Non Personal	43517	1247	49
Actual Value	*Personal*	2011	4911	17
	Sensitive	55	80	170

Non linear SVMs use a radial basis function kernel. By using random search, we find that the model works best with 100 PCA components and n-lines of 1-2. However the model has very poor results as shown in Table 5.

Table 5: Non Linear SVM Line By Line Results

	Non Personal	*Personal*	*Sensitive*
F_2 **Scores**	0.99	0.029	0

4.5 Gradient Boosting

Gradient Boosting is chosen as a model to explore due to the algorithm's handling of overfitting, unbalanced data, and computational time. Our data, as stated above, is highly unbalanced in certain areas. Therefore, we use extreme gradient boosting to try and minimize this. The testing shows this algorithm works best with the combination of Word Count Vectorization, TF-IDF, and PCA with twenty components as the preprocessing techniques. With other models higher PCA components often leads to higher accuracy, however this is not the case with extreme gradient boosting. For example, when the components are increased from twenty to two hundred, the computational time dramatically increases with no higher accuracy. Extreme gradient boosting produces the results found in Table 6.

Table 6: Extreme Gradient Boosting F_2 Score - Line by Line Results

	Non Personal	*Personal*	*Sensitive*
F_2 **Scores**	0.960	0.582	0.571

		Predictions		
	Confusion Matrix	*Non Personal*	*Personal*	*Sensitive*
	Non Personal	14934	446	2
Actual Value	*Personal*	1369	1659	7
	Sensitive	36	4	45

4.6 Neural Networks

Neural networks prove to be quite successful for this problem. When being trained on document-level labels, neural networks consistently achieve about 90% accuracy (with similar F_2 scores). We use neural networks in conjunction with Word Count Vectorization, TF-IDF, and PCA with 430 components. We use 430 principal components because this number explains over 90% of the variance. Additionally, testing various numbers of components shows that above this number, the model does not perform any better given more principal components.

To find the best configuration of the neural network, we systematically tune every hyperparameter for each layer one at a time and test before adding new layers. First, we test the optimal number of units in the first hidden layer. The resultant F_2 scores are shown in Figure 17.

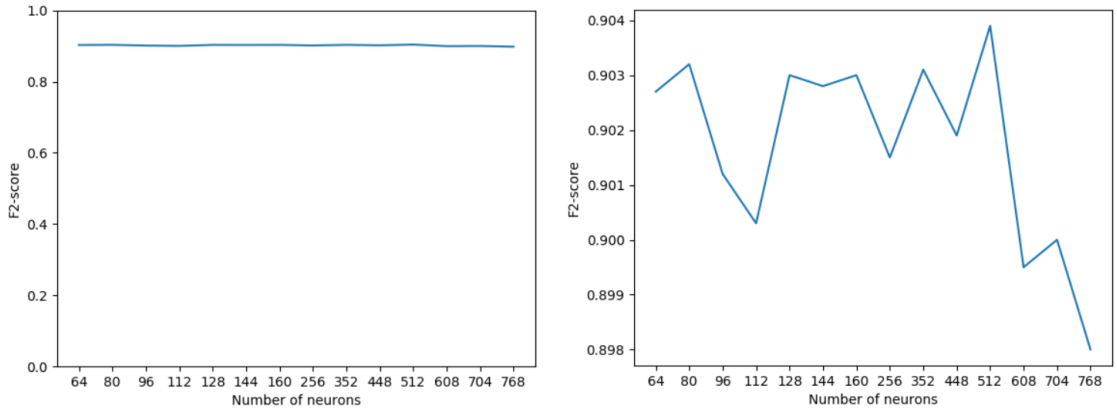


Figure 17: Effect of different numbers of neurons in the first layer on the resultant F_2 score

As one can see, all of the F_2 scores are within a small range of each other (within 0.006), which suggests that choices in this range may not have a large effect. It's possible that the fluctuations between 64 and 512 are random. Neuron ranges below 64 and above 786 perform significantly worse.

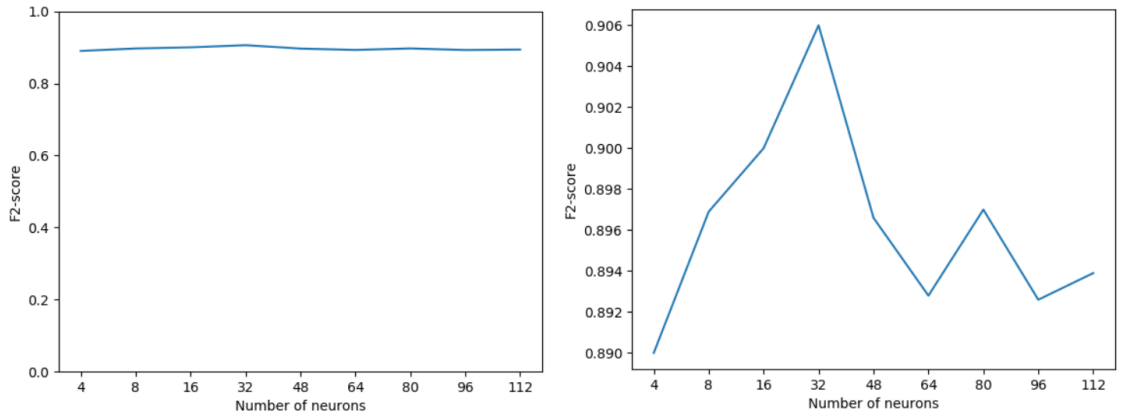


Figure 18: Effect of different numbers of neurons in the second layer on the resultant F_2 score

Adding a second layer creates more of a difference between numbers of units (shown in Figure 18), but do not increase the F_2 score by a very large amount at best. However, one benefit is that using more than one layer reduces the number of training epochs required to reach convergence from over 100 epochs down to about 40, resulting in faster training times. A higher number of layers have little effect on both the F_2 scores and training times after two layers. It is possible that just one or two layers is sufficient to completely estimate the underlying trends in our current dataset, and that more data would be required in order to use more powerful network configurations without overfitting.

Additionally, all of the above configurations are tested with dropout applied. As expected, dropout increases our ability to use more complex models without overfitting. We also test the network with a third and fourth layer, testing different number of units and amount of dropout. After searching through the configurations manually we conduct random search with 300 iterations. The top ten configurations from this search all have F_2 scores within .002 of the others, which indicates that the complex configurations are equally as good as the faster, less complex models. We choose to use a network configuration with 128 units in the first hidden layer and 32 units in the second hidden layer, along with dropout of 0.25 applied to each hidden layer in the final model. For each of the hidden layers, we use the ReLU activation function. We choose to use ReLU activation function because it is non-linear and extremely efficient to compute. While ReLU is not continuously differentiable, this is not a problem in practice. By using the softmax function for activation, the final layer in the network represents a dependent probability distribution over the three possible outcomes–Non-Personal, Personal, and sensitive Personal. Because a sensitive Personal document can still contain Personal information, one could assume that the probabilities for each privacy category are independent and should not necessarily sum to one. In practice however, a dependence assumption resulted in better performance and enables usage of the confidence metric discussed in section 3.4. After softmax is computed, the category with the highest probability is chosen. The above examples come from training the networks using document labels to predict document labels. We also train neural networks on the line level to predict line labels. Table 7 shows the result of this testing.

Table 7: Neural Network F_2 Score - Binary Line by Line Results

	Non Personal	*PII*
F_2 **Scores**	0.79	0.83

		Predictions	
	Confusion Matrix	*Non Personal*	*PII*
Actual Value	*Non Personal*	3990	308
	PII	1250	2451

Table 8 shows the F_2 scores for each category for a neural network trained on document labels.

Table 8: Result of neural network trained on documents

	Non Personal	*Personal*	*Sensitive*
F_2 **Scores**	0.909	0.895	0.896

		Predictions		
	Confusion Matrix	*Non Personal*	*Personal*	*Sensitive*
	Non Personal	93	4	9
Actual Value	*Personal*	3	98	12
	Sensitive	0	1	36

5 Conclusion & Future Work

As seen in Section 4, it is shown that Feed Forward Neural Networks outperform other models when applied to the problem of GDPR classification. When tested on a vault of data this model had final F_2 scores of 0.93 for Non-Personal data, 0.76 for Personal data, and 0.82 for sensitive Personal data on the document level. On the line level the model achieved F_2 scores of 0.95 for Non-Personal data and 0.74 for Personal and sensitive Personal data (See Appendix B for the confusion matrices). These final accuracies have the potential of being unrepresentative for all data because our dataset is constructed from documents compiled from one source. Therefore the model classifies certain documents very well, but it may not perform as well on documents from different sources or of different forms. However, the final system will still help decrease the amount of manual workload needed for classifying documents exponentially.

Our project is successful at aiding in the classification of documents based on the PII contained within them. However, our team believes there is more to be expanded upon. The first of these expanded implementations is the optical character recognition. Certain documents that are converted to text are misconstrued because of the OCR, which in turn alters or removes PII. This can lead to the misclassification of documents. Furthermore, we believe that the accuracy of the system could be improved if there were more training data. Due to time and access to data, our team was only able to classify a certain threshold of documents. With greater manpower and a larger dataset to work with, more documents could be pre-classified within a training set using our HTML tag system and placing documents within their respective GDPR PII categories. With a greater corpus of training data available, it's possible that certain preprocessing techniques such as Word2Vec may become more effective. It was hypothesized in Section 4.1 that Word2Vec failed due to the relatively small training dataset. This problem may be solved if our recommendations are executed.

References

[1] What does tf-idf mean, 2018. URL http://www.tfidf.com/.

[2] European Union. Eu gdpr, 2018. URL https://eugdpr.org/.

[3] European Commission. Rules for business and organisations, 2018. URL https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations_en.

[4] European Commission. 2018 reform of eu data protection rules, 2018. URL https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en.

[5] Randy Paffenroth. Statistical learning class notes, 2018.

[6] Anup Bhande. What is underfitting and overfitting in machine learning and how to deal with it., 2018. URL https://cdn-images-1.medium.com/max/1200/1*_7OPgojau8hkiPUiHoGK_w.png.

[7] Russell Brown. More nlp with sklearn's countvectorizer, 2017. URL https://medium.com/@rnbrown/more-nlp-with-sklearns-countvectorizer-add577a0b8c8.

[8] Jeffrey Dean et al. Distributed representation of words and phrases and their compositionality, 2013. URL https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[9] Sebastian Ruder. On word embeddings - part 3: The secret ingredients of word2vec, 2016. URL http://ruder.io/secret-word2vec/.

[10] Mubaris NK. Word embedding - word2vec, 2017. URL https://mubaris.com/2017/12/14/word2vec/.

[11] Victor Powell. Principal component analysis. URL http://setosa.io/ev/principal-component-analysis/.

[12] Steven M. Holland. Principal components analysis (pca), May 2008. URL https://strata.uga.edu/software/pdf/pcaTutorial.pdf.

[13] Ruye Wang. Normal, hermitian, and unitary matrices, 2015. URL http://fourier.eng.hmc.edu/e161/lectures/algebra/node4.html.

[14] Andrew Ng et al. Unsupervised feature learning and deep learning, 2015.

[15] Michael A. Nielsen. Neural networks and deep learning, 2015.

[16] Dan-Ching Liu. A practical guide to relu, 2017. URL https://medium.com/tinymind/a-practical-guide-to-relu-b83ca804f1f7.

[17] Avinash Sharma V. Understanding activation functions in neural networks, 2017. URL https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[18] Multi-class neural networks: Softmax, 2018. URL https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax.

[19] Geoffrey Hinton et al. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 2014.

[20] Sunil Ray. 6 easy steps to learn naive bayes algorithm, 2017. URL https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/.

[21] Alex Rogozhnikov. Gradient boosting explained, 2016. URL http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html.

[22] Niklas Donges. The random forest algorithm, 2018. URL https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd.

[23] Gareth James et al. *An Introduction to Statistical Learning*. Springer, 2017.

[24] Harish Kandan. Understanding the kernel trick, 2017. URL https://towardsdatascience.com/understanding-the-kernel-trick-e0bc6112ef78.

[25] Ming-Husan Yang. Introduction to support vector machines. URL http://u.cs.biu.ac.il/~haimga/Teaching/AI/saritLectures/svm.pdf.

[26] R. Berwick. An idiot's guide to support vector machines. URL http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf.

[27] Reducuing loss: Gradient descent, 2018. URL https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent.

[28] Walber. Precision and recall, 2014. URL https://commons.wikimedia.org/wiki/File:Precisionrecall.svg.

[29] Parbhu. Understanding hyperparameters and its optimisation techniques, 2018. URL https://towardsdatascience.com/understanding-hyperparameters-and-its-optimisation-techniques-f0debba07568.

[30] Deniz Yuret. Machine learning in 10 pictures, 2014. URL http://www.denizyuret.com/2014/02/machine-learning-in-5-pictures.html.

[31] Detlev Gabel & Tim Hickman. Chapter 5: Key definitions - unlocking the eu general data protection regulation, 2017. URL https://www.whitecase.com/publications/article/chapter-5-key-definitions-unlocking-eu-general-data-protection-regulation.

[32] European Commission. What does the general protection regulation (gdpr) govern?, 2018. URL https://ec.europa.eu/info/law/law-topic/data-protection/reform/what-does-general-data-protection-regulation-gdpr-govern_en.

[33] Gerard Nico. Train test error, 2018. URL https://gerardnico.com/data_mining/overfitting.

[34] Matt Brems. A one-stop shop for principal component analysis, 2017. URL https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c.

[35] Neeraj Singh Sarwan. Intuitive understanding of word embeddings, 2017. URL https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/.

[36] Prince Gover. Gradient boosting from scratch, 2017. URL https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d.

[37] What is the difference between bagging and boosting, 2016. URL https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/.

A Labeling Examples

Multi-line Case:
<location><data>Street
City, State
Zip Code</data></location>

Multiple Data tags related in a line:
<name_phone><data>Name: John Smith | Phone Number: +1(111)1111</data></name_phone>

B Vault Confusion Matrices

Table 9: Results of neural network trained on documents- Vault

		Predictions		
	Confusion Matrix	$Non-Personal$	$Personal$	$Sensitive$
	$Non-Personal$ 30	1	1	
Actual Value	$Personal$	3	19	4
	$Sensitive$	0	1	8

Table 10: Results of neural network trained on lines - Vault

		Predictions	
	Confusion Matrix	$Non-Personal$	$Personal/Sensitive$
Actual Value	$Non-Personal$	7295	403
	$Personal/Sensitive$	491	1365