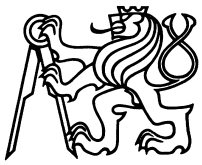
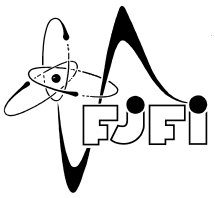


Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering



Active Learning for Text Classification

Aktivní učení pro klasifikaci textů

Masters's Degree Project

Author: **Marko Sahan**

Supervisor: **doc. Ing. Václav Šmídl, Ph.D.**

Academic year: 2019/2020

- Zadání práce -

- Zadání práce (zadní strana) -

Acknowledgment:
I would like to thank ..... for (his/her expert guidance) and express my gratitude to ..... for (his/her language assistance).

Author's declaration:
I declare that this Masters's Degree Project is entirely my own work and I have listed all the used sources in the bibliography.

Prague, July 8, 2019 Jméno Autora

Název práce:
Aktivní učení pro klasifikaci textů

Autor: Marko Sahan

Obor: Aplikované matematicko-stochastické metody

Druh práce: Diplomová práce

Vedoucí práce: **doc. Ing. Václav Šmídl, Ph.D.**, Ústav teorie informace a automatizace

Abstrakt:

Klíčová slova:

Title:
Active learning for text classification

Author: Marko Sahan

Abstract:

Key words:

Contents

Introduction

Chapter 1

Introduction to Decision Theory

Easily speaking, decision can be defined as a set of actions. We are going to work with text data and solve text classification problem. Thus, our main purpose is to introduce precise mathematical abstractions for an automatic and optimal decision making.

Considering binary classification problem. We are seeking to find such set of actions that will assign each input value to its class.

We would like to commence our formal definition with the data. Let $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^n$ and $\mathbf{y} \in \mathcal{Y} = \{(0,1)^T, (1,0)^T\}$, where $\mathbf{x}$ are feature vectors of size n and $\mathbf{y}$ are its labels assigned to the data from space $\mathcal{X}$. Each value from space $\mathcal{Y}$ can be represented as a one hot representation that consist from ones and zeros $\mathbf{y} \in \{(0,1)^T, (1,0)^T\}$. Therefore, first class is represented as $\mathbf{y} = (1,0)^T$ and second class is represented as $\mathbf{y} = (0,1)^T$. As a good example of previous definition, $\mathbf{x}$ can be a text document (represented in a mathematical form in order to meet a definition above) and $\mathbf{y}$ can be its label that will show if the document is relevant or not. As seen from an example, label and text are forming a tuple. In this work we are also considering our data as tuples of variables $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$.

1.1 Decision Theory

In this section we would like to define solution of a classification problem as finding such set of actions (decision) that minimizes loss. By loss we can understand classification error, entropy, etc.. This problem is still set up vaguely because neither decision nor loss was properly defined.

Each classification problem can be summarized as finding the optimal boundary that will split the dataset with respect to labels. The boundary can be defined as an action that is done with respect to some conditions. Let $a \in \mathcal{A}$ is an action and $\mathcal{A}$ is an action space. As mentioned previously, we would like to make a decision (action) that will split the dataset in two sets and we want this classification to be as good as possible. Quality of the classification can be measured with specific metrics which will be introduced in further sections. In our case we would like to minimize losses (error of classification). As a result, tandem of an action and loss minimization metric lets us to understand how good the action is. Subsequent step is an introduction of a loss function L. Loss function is dependent on action $a \in \mathcal{A}$. Formal definition of the loss function is shown next.

According to [2] loss function L can be defines as

$$L = L(\theta, a), \tag{1.1}$$

where $\theta \in \Theta$ is parameters' vector where Θ is parameters space and $a \in \mathcal{A}$ is an action where $\mathcal{A}$ is an action space. In [2], it is also said that parameter θ can be understand as the true state of nature.

Definition above shows us loss with respect to one action and one true state of nature. Thus, we would like to define expected loss function.

Definition 1. [2]If $\pi^*(\theta)$ is believed probability distribution of θ at the time of decision making, the *Bayesian expected loss* of an action a is

$$\rho(\pi^*, a) = \mathbb{E}_{\pi^*}[L(\theta, a)], \tag{1.2}$$

$$= \int_{\Theta} L(\theta, a)\pi^* d\theta. \tag{1.3}$$

Basing on definition 1.2, the optimal action is the one that minimizes expected loss, which is defined as

$$a^* = \underset{a \in \mathcal{A}}{\operatorname{argmin}} \mathbb{E}_{\pi^*}[L(\theta, a)]. \tag{1.4}$$

However, how can we connect optimal action a^* with given data? Basing on the data definitions from previous part, we can assume that $\mathcal{X} \times \mathcal{Y}$ is an infinite set and $(\mathbf{x}, \mathbf{y})$ are samples from this set. Considering $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$ are random variables. If both $\mathbf{x}$ and $\mathbf{y}$ are random variables then tuple $(\mathbf{x}, \mathbf{y})$ is a sample from joint probability density function $p(\mathbf{x}, \mathbf{y})$. With the usage of conditional probability rule $p(\mathbf{x}, \mathbf{y})$ can be written as

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x}). \tag{1.5}$$

At this part we are able to show which meaning this optimal action a^* has. In equation 1.5 $p(\mathbf{x})$ is given and $p(\mathbf{y}|\mathbf{x})$ is an unknown pdf that we want to estimate. The optimal action will lead us to an estimate of $p(\mathbf{y}|\mathbf{x})$.

1.2 Decision Theory for Supervised Learning

Supervised learning requires validation set of the data. Validation set is splitted on training and testing sets. Thus, we have to extend data definition.

Considering the data $\tilde{\mathbf{X}} \subset \mathcal{X}$ and its labels $\tilde{\mathbf{Y}} \subset \mathcal{Y}$. We define cartesian product $\tilde{\mathbf{X}} \times \tilde{\mathbf{Y}}$ as a validation set.

1.2.1 Decision Theory and Support Vector Machine Algorithm

In this subsection we will continue construction of the decision theory on the example of Support Vector Machine (SVM) method. For simplicity lets consider linearly separable dataset. From the theoretical perspective SVM constructs hyperplane in high dimensional space that separates two classes. In this case our decision (action) is a hyperplane that will separate two classes. Equation of the hyperplane can be written as $f(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}^T\mathbf{x} + b$ where $\mathbf{w} \in \mathbb{R}^n$ is a set of hyperplane parameters and $b \in \mathbb{R}$ is a bias . As a result, action space is represented as $(\mathbb{R}^n, \mathbb{R}) = \mathcal{A}$ and as a consequence tuple $(\mathbf{w}, b) \in \mathcal{A}$. From this knowledge we can define $\theta = (\mathbf{x}, \mathbf{y})$ where tuple $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$ is parameters and $\Theta = \mathcal{X} \times \mathcal{Y}$ is a parameters' space. However, for now we are limited on $\Theta = \tilde{\mathbf{X}} \times \tilde{\mathbf{Y}}$. Considering loss function 1.1 that can be written as

$$L = L(\mathbf{x}, \mathbf{y}, \mathbf{w}, b). \tag{1.6}$$

Following task is to understand how good is our action (hyperplane estimation) with respect to the dataset. We can choose different types of loss functions such as cross entropy, hinge loss, etc.. The most basic approach for SVM method is hinge loss function [7] which is defined as

$$L(\mathbf{x}, \mathbf{y}, \mathbf{w}, b) = \max(0, 1 - y\hat{y}(\mathbf{x}, \mathbf{w}, b)) \tag{1.7}$$

where $\hat{y}(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}^T\mathbf{x} + b$ and $y = \mathbf{y}_1$.

In terms of SVM method we want to find such hyperplane that will label input values as a first class if it is "above" the hyperplane and as a second class if it is "below" the hyperplane. At this point very important assumption will be introduced. In order to find an optimal hyperplane we assume that the data $\tilde{\mathbf{X}}$ and its labels $\tilde{\mathbf{Y}}$ fully describe spaces $\mathcal{X}$ and $\mathcal{Y}$. Moreover, as mentioned earlier, we consider $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$ are random variables with joint probability density function $p(\mathbf{x}, \mathbf{y})$. We will also assume that $\forall i \in \{1, .., N\}$, $(\mathbf{x}_i, \mathbf{y}_i) \in \tilde{\mathbf{X}} \times \tilde{\mathbf{Y}}$ are independent identically distributed. As a result, with the usage of those data, probability density function $p(\mathbf{x}, \mathbf{y})$ can be estimated as

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{x} - \mathbf{x}_i, \mathbf{y} - \mathbf{y}_i) \tag{1.8}$$

where $\delta(\mathbf{x} - \mathbf{x}_i, \mathbf{y} - \mathbf{y}_i)$ is Dirac delta function which is centered in $(\mathbf{x}_i, \mathbf{y}_i)$.

Using (1.2) we can evaluate expected loss function for SVM as follows

$$\begin{aligned}
\mathbb{E}_{\pi^*} L &= \int_{\mathcal{X}\times\mathcal{Y}} L(\mathbf{x}, \mathbf{y}, \mathbf{w}, b) p(\mathbf{x}, \mathbf{y}) d(\mathbf{x}, \mathbf{y}), \\
&= \int_{\mathcal{X}\times\mathcal{Y}} \max(0, 1 - y_1 \hat{y}(\mathbf{x}, \mathbf{w}, b)) \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{x} - \mathbf{x}_i, \mathbf{y} - \mathbf{y}_i) d(\mathbf{x}, \mathbf{y}), \\
&= \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_{1,i} \hat{y}(\mathbf{x}_i, \mathbf{w}, b))
\end{aligned}$$

where $\hat{y}(\mathbf{x}_i, \mathbf{w}, b) = \mathbf{w}^T\mathbf{x}_i + b$ and $y_{1,i}$ is first component of $i-$ th vector $\mathbf{y}_i$ Expect loss function for SVM can be written as

$$\rho(\mathbf{x}_i, \mathbf{w}, b) = \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)). \tag{1.9}$$

1.2.2 Decision Theory and Algorithm Based on Neural Network Function

Decision Theory construction for the algorithm, based on a neural network function, is mostly the same as in 1.2.1. However in this case our decision is to find estimate $\hat{\mathbf{y}} = \hat{\mathbf{y}}(\mathbf{x}, \mathbf{W}, \mathbf{b})$ of the probability density function $p(\mathbf{y}|\mathbf{x})$ where $\mathbf{x}$ is the input data, $\mathbf{W}$ is a set of all neural network function parameters and $\mathbf{b}$ is a vector of biases. Action space $\mathcal{A}$ will be parameters' and biases' space of $\hat{\mathbf{y}}$. Same as in 1.2.1 we can define $(\mathbf{x}, \mathbf{y})$ are parameters of the loss function and $\mathcal{X} \times \mathcal{Y}$ is a parameters' space. Another example of loss functions that we will use is cross entropy loss function, which is defined as

$$L(\mathbf{x}, \mathbf{y}, \mathbf{W}, \mathbf{b}) = -y_1 \ln(\hat{y}_1(\mathbf{x}, \mathbf{W}, \mathbf{b})) - y_2 \ln((\hat{y}_2(\mathbf{x}, \mathbf{W}, \mathbf{b})), \tag{1.10}$$

where $\mathbf{y} = (y_1, y_2)^T$ **and** $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2)^T$**.** We consider $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$ are random variables with joint probability density function $p(\mathbf{x}, \mathbf{y})$. With the usage of the given dataset where $\forall i \in \{1, .., N\}$, $(\mathbf{x}_i, \mathbf{y}_i) \in \tilde{\mathbf{X}} \times \tilde{\mathbf{Y}}$ are independent identically distributed we can approximate $p(\mathbf{x}, \mathbf{y})$ as (1.8). Applying definition

1, expected loss for the algorithm based on a neural network function is evaluated as

$$\begin{aligned}\mathbb{E}_{\pi^*}L &= \int_{\mathcal{X}\times\mathcal{Y}} L(\mathbf{x}, y, \mathbf{W}, \mathbf{b})p(\mathbf{x}, \mathbf{y})d(\mathbf{x}, \mathbf{y}),\\ &= -\int_{\mathcal{X}\times\mathcal{Y}} \Big(y_1 \ln\left(\hat{y}_1(\mathbf{x}, \mathbf{W}, \mathbf{b})\right) + y_2 \ln\left((\hat{y}_2(\mathbf{x}, \mathbf{W}, \mathbf{b}))\right)\Big)\frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{x}-\mathbf{x}_i, \mathbf{y}-\mathbf{y}_i)d(\mathbf{x}, \mathbf{y}),\\ &= -\frac{1}{N}\sum_{i=1}^{N}\Big(y_1 \ln\left(\hat{y}_1(\mathbf{x}, \mathbf{W}, \mathbf{b})\right) + y_2 \ln\left((\hat{y}_2(\mathbf{x}, \mathbf{W}, \mathbf{b}))\right)\Big), \qquad (1.11)\end{aligned}$$

where $\mathbf{y} = (y_1, y_2)^T$ **and** $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2)^T$**.**

1.2.3 Decision Theory and Naive Bayes Algorithm

Naive Bayes algorithm is a bit different to the algorithm based on Neural Networks and SVM. In the case of Naive Bayes we want to estimate $p(\mathbf{W}|\mathbf{x}, \mathbf{y})$ where $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_n) \subset \mathcal{W}$ is an action ($a = \mathbf{W},\ a \in \mathcal{A}$). The reason why we look for an estimate of the $p(\mathbf{W}|\mathbf{x}, \mathbf{y})$ but not $p(\mathbf{y}|\mathbf{x}, \mathbf{w})$ is due to the fact that in case of estimating $p(\mathbf{y}|\mathbf{x}, \mathbf{W})$ we would have to work with normalization constant would be dependent on the set of parameters $\mathbf{W}$. That fact would make our computations very complicated. Before continuing with a loss function construction we would like to go trough Naive Bayes (NB) method.

1.2.3.1 Naive Bayes

Assuming binary classification problem. With the usage of Bayes rule we can rewrite $p(\mathbf{W}|\mathbf{x}, \mathbf{y})$ as follows

$$p(\mathbf{W}|\mathbf{x}, \mathbf{y}) = \frac{p(\mathbf{y})p(\mathbf{x}|\mathbf{y}, \mathbf{W})p(\mathbf{W})}{\int_{\mathcal{W}} p(\mathbf{x}, \mathbf{y}|\mathbf{W})p(\mathbf{W})d\mathbf{W}} \qquad (1.12)$$

where $\mathcal{W}$ is assumed as a space of $\mathbf{W}$.

Naive Bayes method introduces very strong assumption in equation 1.12. This assumption says that features of vector $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ are conditionally independent. As a result estimation of $p(\mathbf{W}|\mathbf{x}, \mathbf{y})$ can be estimated as

$$\tilde{p}(\mathbf{W}|\mathbf{x}, \mathbf{y}) = \frac{1}{Z}p(\mathbf{y})p(\mathbf{W})\prod_{i=1}^{n}\left(p(x_i|y_1, \mathbf{w}_i)^{y_1}p(x_i|y_2, \mathbf{w}_i)^{y_2}\right), \qquad (1.13)$$

where $\mathbf{y} = (y_1, y_2)$, $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_n)$, Z is a normalizing constant.

1.2.3.2 Decision Theory

We want to maximize probability $\tilde{p}(\mathbf{W}|\mathbf{x}, \mathbf{y})$. As a result, using 1.13, loss function L will be represented as

$$L(\mathbf{y}, \mathbf{x}, \mathbf{w}) = -\log\left(\tilde{p}(\mathbf{W}|\mathbf{x}, \mathbf{y}\right), \qquad (1.14)$$

$$= \log(Z) - \log\left(p(\mathbf{y})\right) - \log\left(p(\mathbf{W})\right) - \sum_{i=1}^{n}\log\left(p(x_i|y_1, \mathbf{w}_i)^{y_1}p(x_i|y_2, \mathbf{w}_i)^{y_2}\right). \qquad (1.15)$$

Same as in 1.2.1 and 1.2.2 we will assume that we can approximate $p(\mathbf{x}, \mathbf{y})$ as 1.8. From this moment everything is ready for deriving expected loss function. Expected loss function for Naive Bayes method

is derived as

$$\mathbb{E}_{\pi^*} L = \int_{\mathcal{X}\times\mathcal{Y}} L(\mathbf{x},\mathbf{y},\mathbf{w}) p(\mathbf{x},\mathbf{y}) d(\mathbf{x},\mathbf{y}),$$
$$\int_{\mathcal{X}\times\mathcal{Y}} \Big(\xi(\mathbf{W},\mathbf{y}) - \sum_{i=1}^{n} \log\left(p(x_i|y_1,\mathbf{w}_i)^{y_1} p(x_i|y_2,\mathbf{w}_i)^{y_2}\right) \frac{1}{N}\sum_{j=1}^{N} \delta(\mathbf{x}-\mathbf{x}_j,\mathbf{y}-\mathbf{y}_j) d(\mathbf{x},\mathbf{y}),$$
$$\frac{1}{N}\sum_{j=1}^{N} \Big(\xi_j(\mathbf{W},\mathbf{y}_j) - \sum_{i=1}^{n} \log\left(p(x_{i,j}|y_{1,j},\mathbf{w}_i)^{y_{1,j}} p(x_{i,j}|y_{2,j},\mathbf{w}_i)^{y_{2,j}}\right)\Big) \tag{1.16}$$

where $\xi(\mathbf{W},\mathbf{y}) = \log(Z) - \log\left(p(\mathbf{y})\right) - \log\left(p(\mathbf{W})\right)$ **and** $\xi_j(\mathbf{W},\mathbf{y}_j) = \log(Z) - \log\left(p(\mathbf{y}_j) - \log\left(p(\mathbf{W})\right)\right)$**.**

1.2.4 Decision Theory and Random Forest Algorithm

In order to work with random forests we must precisely define decision trees and only then construct theory for random forests.

1.2.4.1 Decision Tree

In this section we expect our decision three to give us an estimate $\hat{\mathbf{y}}(\mathbf{x},\mathbf{w}) \in \{(0,1)^T,(1,0)^T\}$ where $\mathbf{w}$ is a vector that describes tree (depth, branches, etc.), $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$. It is very important to mention that for different trees $\mathbf{w}$ can have different dimensionality. Thus, for consistency we will assume that for all $\mathbf{w} \in \mathcal{W}$ exists upper bound, where $\mathcal{W}$ is redefined as a space of tree parameters. As a result we will make all $\mathbf{w}$ same length. If $\mathbf{w}$ has spare elements, they will be filled with zeros. Parameter space will be same as in 1.2.1-1.2.3, whereas action $a \in \mathcal{A}$ will be represented as $\mathcal{A} = \mathcal{W}$. In order to understand when our tree is optimal we can use zero-one loss function. Zero-one loss function is defined as

$$L(\mathbf{x},\mathbf{y},\mathbf{w}) = \begin{cases} 1, & \mathbf{y} \neq \hat{\mathbf{y}}(\mathbf{x},\mathbf{w}) \\ 0, & \mathbf{y} = \hat{\mathbf{y}}(\mathbf{x},\mathbf{w}) \end{cases}. \tag{1.17}$$

With the usage of the given data where $\forall i \in \{1,..,N\}$, $(\mathbf{x}_i,\mathbf{y}_i) \in \tilde{\mathbf{X}} \times \tilde{\mathbf{Y}}$ are independent identically distributed we can approximate $p(\mathbf{x},\mathbf{y})$ as (1.8). As a result, expected loss function for decision tree can be derived as

$$\begin{aligned} \mathbb{E}_{\pi^*} L &= \int_{\mathcal{X}\times\mathcal{Y}} L(\mathbf{x},\mathbf{y},\mathbf{w}) p(\mathbf{x},\mathbf{y}) d(\mathbf{x},\mathbf{y}), \\ &= \int_{\mathcal{X}\times\mathcal{Y}} L(\mathbf{x},\mathbf{y},\mathbf{w}) \frac{1}{N}\sum_{i=1}^{N} \delta(\mathbf{x}-\mathbf{x}_i,\mathbf{y}-\mathbf{y}_i) d(\mathbf{x},\mathbf{y}), \\ &= \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{x}_i,\mathbf{y}_i,\mathbf{w}) \end{aligned}$$

where $\sum_{i=1}^{N} L(\mathbf{x}_i,\mathbf{y}_i,\mathbf{w})$ is (1.17).

If we want to minimize expected loss we have to follow next steps. While construction decision tree we choose such feature $x_i \in (x_1,...,x_n)^T = \mathbf{x}$ that will bring as the highest information about the system. This feature will form first layer, then we add another feature with the highest informational gain and construct second layer. Basing of this method we construct nodes and add more and more layers (branches).

In the following part we are going to work with a set of decision trees. For this purposes we will define our decision tree as $\hat{\mathbf{y}} = \left(T_1(\mathbf{x},\mathbf{w}_l), T_2(\mathbf{x},\mathbf{w}_l)\right)^T$ where index l represents set of parameters for $l-$th three and indices $1,2$ represent first and second value of the one-hot vector.

1.2.4.2 Random Forest

Considering $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$ as random variables with joint probability density function $p(\mathbf{x}, \mathbf{y})$. We will also assume that $\forall i \in \{1, .., N\}$, $(\mathbf{x}_i, \mathbf{y}_i) \in \tilde{\mathbf{X}} \times \tilde{\mathbf{Y}}$ are independent identically distributed.

With the usage of $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ we will make $\{1, ..., L\}$, $L \in \mathbb{N}$ sets where $\forall l \in L$, $\tilde{\mathbf{X}}_l \subset \tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}_l \subset \tilde{\mathbf{Y}}$. The data $\tilde{\mathbf{X}}_l$ and $\tilde{\mathbf{Y}}_l$ are created with random uniform sampling from $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$. We also want each subset to contain strictly 60% of the data from $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$. As a result parameter space for random forests will form tuples of sets $(\tilde{\mathbf{X}}_l, \tilde{\mathbf{Y}}_l)$. Basing on this theory we will construct L decision trees $\hat{y} = T(\mathbf{x}, \mathbf{w}_l)$, where $\mathbf{x} \in \tilde{\mathbf{X}}_l$. As a result for $l-$ th decision tree

$$\mathbb{E}_{\pi^*} L = \frac{1}{N_l} \sum_{i=1}^{N_l} L(\mathbf{x}_{i,l}, \mathbf{y}_{i,l}, \mathbf{w}_l) \delta(\mathbf{x} - \mathbf{x}_{i,l}, \mathbf{y} - \mathbf{y}_{i,l}) \tag{1.18}$$

where $(\mathbf{x}_{i,l}, \mathbf{y}_{i,l}) \in (\tilde{\mathbf{X}}_l, \tilde{\mathbf{Y}}_l)$ and N_l is a number of the data in $\tilde{\mathbf{X}}_l$ and $\tilde{\mathbf{Y}}_l$. If we assume $\mathbf{w}_l$ as a random variable then L decision trees form samples from probability density function $p(\mathrm{y}|\mathbf{x}, \mathbf{w})$. In other words

$$p(\mathbf{y}|\mathbf{x}, \mathbf{w}_l) = T_1(\mathbf{x}, \mathbf{w}_l)^{y_1} T_2(\mathbf{x}, \mathbf{w}_l)^{y_2} \tag{1.19}$$

where label $\mathbf{y}$ is written as a one-hot representation $\mathbf{y} = (y_1, y_2)^T$. Thus, we can say that classification probability $p(\mathbf{y}|\mathbf{x})$ can be written as

$$p(\mathbf{y}|\mathbf{x}) = \int_{\mathbf{w} \in \mathcal{A}} p(\mathbf{y}|\mathbf{x}, \mathbf{w}) p(\mathbf{w}) d\mathbf{w}. \tag{1.20}$$

where $\mathcal{A}$ is an action space. With the usage of samples $\mathbf{w}_l$ we can approximate $p(\mathbf{y}|\mathbf{x})$ as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{L} \sum_{l=1}^{L} T_1(\mathbf{x}, \mathbf{w}_l)^{y_1} T_2(\mathbf{x}, \mathbf{w}_l)^{y_2} \tag{1.21}$$

where each decision tree $(T_1(\mathbf{x}, \mathbf{w}_l), T_2(\mathbf{x}, \mathbf{w}_l))^T$ is constructed with the usage of (1.18).

Before continuing with further sections we define output of Random Forest as $\hat{\mathbf{y}} = \hat{\mathbf{y}}(\mathbf{x}, \mathbf{W})$, where $\mathbf{W} = (\mathbf{w}_1, ..., \mathbf{w}_L)$ is set of parameters of specific Random Forest algorithm. We define vector $\hat{\mathbf{y}}$ as

$$\hat{\mathbf{y}} = \frac{1}{L} \sum_{l=1}^{L} (T_1(\mathbf{x}, \mathbf{w}_l), T_2(\mathbf{x}, \mathbf{w}_l)). \tag{1.22}$$

1.3 Decision Theory for Active Learning

As mentioned in previous sections $\tilde{\mathbf{X}} \times \tilde{\mathbf{Y}}$ is a validation set. When we finish a model training, we may think that we need more training data. Thus, we can choose the data from $\mathbf{X} \subset \mathcal{X} \backslash \tilde{\mathbf{X}}$. However it is important to understand that we have no labels for the set $\mathbf{X}$. We can ask for a help from an annotator that can give us those labels. We assume that getting labels needs some time and is very expensive.

Active learning problem is defined as a sequence of Supervised learning problems. Specifically, we assume that labels $\mathbf{y} \in \tilde{\mathbf{Y}}$ are available only for $\mathbf{x} \in \tilde{\mathbf{X}}$. We have the possibility to select an unlabeled element from $\mathbf{X}$ and ask for its label. Since it is expensive, we aim to have such questions that will maximize scores as fast as possible. Formally, we denote $J_0 = \{j_{01}, j_{02}, \dots j_{0N}\} = \{1, \dots, N\}$ the initial set of N available labels. using only the labeled data, the supervised learning task is defined on sets $\mathbf{X}_0 = \tilde{\mathbf{X}} = \{\mathbf{x}_i\}_{i \in J_0}$ and $\mathbf{Y}_0 = \tilde{\mathbf{Y}} = \{\mathbf{y}_i\}_{i \in J_0}$. This set is sequentially extended with new labels gained from

$\mathbf{X}$. We consider a sequence of U questions $u = \{1, \dots, U\}$, in each question, we select an index j_u and ask to obtain the label $\mathbf{y}_{j_u}$ for data record $\mathbf{x}_{j_u}$. The index set and the data sets are extended as follows

$$J_u = \{J_{u-1}, j_u\}, \qquad \mathbf{X}_u = \{\mathbf{X}_{u-1}, \mathbf{x}_{j_u}\}, \qquad \mathbf{Y}_u = \{\mathbf{Y}_{u-1}, \mathbf{y}_{j_u}\}.$$

The task of active learning is to optimize the selection of indices j_u to reach as good classification metrics with as low number of questions as possible. As a result we have to define expected loss for each question u that will be dependent on the action and parameter spaces. In this case we can define our action space $\mathcal{A}_u$ as a space of the data indices with respect to the parameters space Θ_u for each question u. Parameters space is defined as a set of possible parameters from a specific model. It is very important to understand that we will need not only one set of parameters but parameters distribution. We need parameters distribution because we will integrate over the parameters space. As an example, if we talk about SVM method, then parameters space for active learning problem will be defined as a set of weights that form a hyperplane. If we talk about the algorithm that is based on a Neural Network function, then parameters space of the active learning problem will form weights from neurons. We wanted to highlight that parameters space will be different for each problem but the idea for each algorithm is same .

As a result our task can be written as

$$j_u^* = \underset{j \in J \setminus J_u}{\operatorname{argmin}}(\mathbb{E}_{\pi_u^*} L^*) \tag{1.23}$$

where $\mathbb{E}_{\pi_u^*} L^*$ is expected loss that is dependent on an action given question u, and J is space of all indices. Expected loss for the active learning problem is defined as

$$\mathbb{E}_{\pi_u^*} L^* = \int_{\Theta_u} L^*(a, \theta) \pi_u^* d\theta \tag{1.24}$$

where $a \in \mathcal{A}_u$ and $\theta \in \Theta_u$ and L^* is a loss function for the active learning problem. Character "$*$" is used only for distinguishing active learning loss from the loss function which is used for different models. We will specify action space because it will be the same for all models that are used in the active learning section. Action space $\mathcal{A}$ is a set of possible indices $j_u \in J_u$ where u is a specific question. Thus, (1.24) can be written as

$$\mathbb{E}_{\pi_u^*} L^* = \int_{\Theta_u} L^*(j_u, \theta) \pi_u^* d\theta. \tag{1.25}$$

Using this approach we will be able sequentially select indices from $\mathbf{X}$ and ask for a label from $\mathbf{Y}$ that will help us to get higher scores faster than in the case of random choice of indices.

1.3.1 Bayesian Approach of Classifiers' Parameters Sampling

Considering that $\mathbf{y} \in \mathcal{Y}$. Let $\hat{\mathbf{y}} = \hat{\mathbf{y}}(\mathbf{x}_{j_u}, \theta_u)$ is an estimate of $\mathbf{y}$. However, in this case output estimate $\hat{\mathbf{y}}$ is represented as a vector of probabilities that $\mathbf{x}_{j_u}$ is assigned to different classes. As an example for well trained binary classifier, for specific $\mathbf{x}$ that is assigned to $\mathbf{y} = (1, 0)^T$, classifiers estimate of $\mathbf{x}$ can be $\hat{\mathbf{y}} = (0.95, 0.05)^T$. It is very interesting that before we can solve the optimization problem with choosing the index j_u we have to solve the optimization problem of finding $\hat{\mathbf{y}}$. This leads us to supervised learning models that we have discussed in previous sections.

In this section we would like to construct theory around π_u^* from equation 1.25. Mentioned distribution is a distribution of the models' parameters given the training data that can be written as

$$\pi_u^* = p_u(\theta_u | \mathbf{X}_u, \mathbf{Y}_u). \tag{1.26}$$

We do not have explicit form of the pdf. However, we assume that we have samples Q_u samples $\theta_{u,q} \in \{1_u, ..., Q_u\}$ from $p_u(\theta_u|\mathbf{X}_u, \mathbf{Y}_u)$. As a result $p_u(\theta_u|\mathbf{X}_u, \mathbf{Y}_u)$ can be approximated as

$$\pi_u^* = \frac{1}{Q_u} \sum_{q=1}^{Q_u} \delta(\theta_u - \theta_{u,q}), \tag{1.27}$$

where $\delta(\theta_u - \theta_{u,q})$ is Dirac delta function centered in $\theta_{u,q}$.

1.3.1.1 Parameters Sampling Based on Training Data Subsets

This method is quite general and can be applied to all types of classifiers in this work (Random Forest, SVM, Neural Network). The idea is very simple. We consider that some data samples in training dataset $\tilde{\mathbf{X}} \times \tilde{\mathbf{Y}}$ are noise corrupted. Thus, its obvious that we do not want our models to learn from noise corrupted data. As a result, we would like to randomly sample Q_u subsets from $\tilde{\mathbf{X}}$ with their labels from $\tilde{\mathbf{Y}}$. Lets rewrite it in more mathematical form.

Assuming N_u is amount of samples in $\mathbf{X}_u$. Let $Z_u = \{z_1, ..., z_{N_u^{sub}}\}$, where $N_u^{sub} \subset N_u$ and $\forall k, l \in N_u^{sub}$, $z_k, z_l \in J_u$, $z_k \neq z_l$. Next step is selection of indices that will form set Z_u. Let $\forall k \in N_u^{sub}$, $z_k \sim U(J_u)$. under condition that we don't want to have duplicate indices in Z_u. Let $\mathbf{X}_{Z_u}$, $\mathbf{Y}_{Z_u}$ are defined as restriction of sets $\mathbf{X}_u$, $\mathbf{Y}_u$ on indices from Z_u. As a result we can approximate 1.26 as

$$\pi_u^* = p_u(\theta_u|\mathbf{X}_{Z_u}, \mathbf{Y}_{Z_u}). \tag{1.28}$$

Sampling from 1.28 is very simple. After training the model using $\mathbf{X}_{Z_u}$ and Y_{Z_u} vector of model parameters will represent a single sample from π_u^*.

1.3.1.2 SGLD

Unlike previous section method, SGLD sampling is designed only for neural networks based classifiers. SGLD modifies Neural Network learning algorithm by adding noise in Stochastic Gradient descent.

1.3.1.3 DENFI

TBD

1.3.2 Active Learning Loss Function

1.3.2.1 Entropy Based Active Learning Loss

First approach of defining Active Learning loss function is negative entropy. Basing of the formal definition of the entropy we can write it as

$$-H(\hat{\mathbf{y}}|\mathbf{x}_{j_u}, \theta_u) = \sum_{r=1}^{R} \hat{y}_r(\mathbf{x}_{j_u}, \theta_u) \log\left(\hat{y}_r(\mathbf{x}_{j_u}, \theta_u)\right), \tag{1.29}$$

where $\hat{y}_r$ is $r-$th element of the output estimate $\hat{\mathbf{y}}$ and θ is a vector of parameters for specific model. As done in Passive Learning sections we want to find expected loss based on entropy function.

With the usage of previous knowledge we can derive expected entropy loss as

$$\begin{aligned}\mathbb{E}_{\pi_u^*} L^* &= \int_{\Theta_u} -H(\hat{\mathbf{y}}|\mathbf{x}_{j_u}, \theta_u) p_u(\theta_u|\mathbf{X}_u, \mathbf{Y}_u) d\theta_u \\ &= \int_{\Theta_u} -H(\hat{\mathbf{y}}|\mathbf{x}_{j_u}, \theta_u) \frac{1}{Q_u} \sum_{q=1}^{Q_u} \delta(\theta_u - \theta_{u,q}) d\theta_u \\ &= \frac{1}{Q_u} \sum_{q=1}^{Q_u} -H(\hat{\mathbf{y}}|\mathbf{x}_{j_u}, \theta_{u,q})) \\ &= \frac{1}{Q_u} \sum_{q=1}^{Q_u} \sum_{r=1}^{R} \hat{y}_r(\mathbf{x}_{j_u}, \theta_{u,q}) \log\left(\hat{y}_r(\mathbf{x}_{j_u}, \theta_{u,q})\right). \end{aligned} \tag{1.30}$$

As a result, minimization of given expected loss will lead us to a sample with the highest entropy.

1.3.2.2 False Positive Sampling Loss

TBD

1.3.3 Active Learning for Random Forests Algorithm

In Supervised Learning section 1.2.4 we have derived that $p(\mathbf{y}|\mathbf{x})$ for Random Forest (RF) algorithm is written as 1.21. Active Learning algorithm requires distribution over the parameters of the Random Forest algorithm. We will solve this problem the way that we will get samples from π_u^* and then approximate probability distribution as (1.26).

In order to estimate samples from π_u^* we define Ensemble Random Forest Algorithm. That means that we will use Q_u Random Forests in each step of Active Learning algorithm. In this case parameters of each $\hat{\mathbf{y}}_{q_u}$, where $\hat{\mathbf{y}}$ is Random Forest one-hot represented output and $q_u \in \{1_u, ..., Q_u\}$, will be iid. Ofcourse RF algorithm is already ensemble of decision trees but in this case we create ensemble algorithm from ensemble algorithms. Previously we defined for each decision tree T_l where $l \in \{1, ..., L\}$, that $\mathbf{w}_l$ is a vector of parameters of $l-$th decision tree. Thus, let $\mathbf{W} = (\mathbf{w}_1, ..., \mathbf{w}_L)$ is set of parameters of specific Random Forest algorithm. Thus, for $\hat{\mathbf{y}}_{q_u}$ we have $\mathbf{W}_{q_u} = (\mathbf{w}_{q_u,1}, ..., \mathbf{w}_{q_u,L})$. As a result we can approximate π_u^* as

$$\pi_u^* = \frac{1}{Q_u} \sum_{q_u=1}^{Q_u} \delta(\mathbf{W} - \mathbf{W}_{q_u}). \tag{1.31}$$

Another important point is to define $\hat{\mathbf{y}}_{q_u} = \hat{\mathbf{y}}(\mathbf{x}_{j_u}, \mathbf{W}_{q_u})$. We formally define each element $\hat{y}_{r,q_u}$ of vector $\hat{\mathbf{y}}_{q_u}$ as

$$\hat{y}_{r,q_u} = \frac{1}{L} \sum_{l=1}^{L} T_r(\mathbf{x}_{j_u}, \mathbf{w}_{q_u,l}),$$

where r is $r-$th class T_r is $r-$th element of a decision tree vector.

With the usage of the theory from section 1.3.2 we can write expected loss for Active Learning Random Forest Algorithm as

$$\mathbb{E}_{\pi_u^*} L_u^* = \frac{1}{Q_u} \sum_{q=1}^{Q_u} \sum_{r=1}^{R} \hat{y}_r(\mathbf{x}_{j_u}, \mathbf{W}_{q_u}) \log\left(\hat{y}_r(\mathbf{x}_{j_u}, \mathbf{W}_{q_u})\right). \tag{1.32}$$

Chapter 2

Natural Language Processing Theory

2.1 Text Representation

According to [3], Natural Language Processing (NLP) is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications.

In this work we are focused on two techniques such as TF-IDF [6] and Fast Text Word Embeddings [5]. These methods are used for representation of text in a mathematical form (vectors, matrices). Even though TF-IDF is quite old method for text representation, it is still widely used. However, primary method, that used in the theses is Fast Text Word Embeddings. In this project we are working with text documents (articles and tweets) and their labels. In the beginning of chapter 1 we defined value $\mathbf{x} \in \mathcal{X}$ as text features vector. By features vector we mean any kind of text encoding (TF-IDF, Fast Text Word Embedding, etc..).

2.1.1 TF-IDF

Term Frequency - Inverse Document Frequency (TF-IDF) is extremely powerful tool. This text encoding tool is quite simple and powerful. Method's advantage is its popularity. Plenty of packages in different programming languages have implementations of this algorithm. As mentioned in the name of this method, it is composed from two parts Term Frequency and Inverse Document Frequency. Term Frequency is defined as

$$TF(t,d) = \frac{f_{t,d}}{\sum_{t'} f_{t',d}},$$

where $f_{t,d}$ is number of times of word t in a document d. Inverse Document Frequency is defined as

$$IDF(t,d) = \log \frac{|D|}{|\{d \in D : t \in d\}|},$$

where numerator stand for total number of documents in the corpus and denominator is number of documents where the term t appears. We are assuming using only words that from corpus D. Thus, the denominator is always greater than zero.

Finally,

$$TF - IDF(t,d) = TF(t,d) \cdot IDF(t,d).$$

N.B

2.1.1.1 TF-IDF and Information Theory

In this part is shown the connection of TF-IDF to Information theory [1]. Lets first take a look on documents' entropy given word t,

$$\begin{aligned} H(D|T=t) &= -\sum_{d} p(d|t)\log p(d|t) \\ &= \log \frac{1}{|\{d \in D : t \in D\}|} \\ &= -\log \frac{|\{d \in D : t \in D\}|}{D|} + \log|D| \\ &= -IDF(t,d) + \log|D|, \end{aligned} \tag{2.1}$$

where D is a documents' random variable and T is words' random variable. Equation 2.1 is correct under condition that we have no duplicate documents in the text corpus. Next step is to derive an equation of mutual information of documents and words as follows

$$\begin{aligned} M(D,T) &= H(D) - H(D|T) \\ &= -\sum_{d} p(d)\log p(d) - \sum H(D|T=t)\cdot p(t)_t \\ &= \sum_{t} p(t)\cdot\Big(\log\frac{1}{|D|} + IDF(t,d) - \log|D|\Big) \\ &= \sum_{t} p(t)\cdot IDF(t,d) \\ &= \sum_{t,d} p(t|d)\cdot p(d)\cdot IDF(t,d) \\ &= \frac{1}{|D|}\sum_{t,d} TF(t,d)\cdot IDF(t,d). \end{aligned} \tag{2.2}$$

As seen from 2.2 TF-IDF has really good explanatory definition based on Information Theory. As a result, it is one more advantage of this method usage. However, here is one big disadvantage that can be very crucial. The higher amount of words is, the bigger and sparser vectors, that represent each document, will be.

2.1.2 Fast Text and CBOW Word Embeddings

Term word embedding means a set of language modeling and feature learning techniques in natural language processing where words or phrases from the vocabulary are mapped to vectors of real numbers. Nowadays exist plenty of word embedding methods based on neural networks and co-occurrence matrices. Word embeddings are used as pretrained models. Words' encoding is used in order to encode the text and then text encoding is used for different purposes such as classification, clustering, etc..

The principle of word embeddings based on neural networks is explained in this section. We decided to describe Continuous Bag of Words Model (CBOW), because Fast Text word embeddings model is a modification of this method and CBOW covers all main theoretical aspects.

2.1.2.1 CBOW Word Embeddings

[4]

We would like to treat text {"The", "cat", 'over", "the', "puddle"} as a context and from these words, be able to predict or generate the center word "jumped". This type of model is a Continuous Bag of Words (CBOW) Model. Before we continue with more theoretical part, it is good to mention that mathematical notation defined here is only used for this section and has no common with the same names of the variables that were defined in the beginning of this theses. First, we want to set up our known parameters. Let the known parameters in our model be the sentence represented by one-hot word vectors. The input one hot vectors or context we will represent with an $\mathbf{x}^{(c)}$. And the output as $\mathbf{y}^{(c)}$ and in the CBOW model, since we only have one output, so we just call this $\mathbf{y}$ which is the one hot vector of the known center word. Now let's define our unknowns in our model. We create two matrices, $\mathcal{V} \in \mathbb{R}^{n \times |V|}$ and $\mathcal{U} \in \mathbb{R}^{|V| \times n}$. Where n is an arbitrary size which defines the size of our embedding space. $\mathcal{V}$ is the input word matrix such that the $i-$th column of $\mathcal{V}$ is the $n-$dimensional embedded vector for word w_i when it is an input to this model. We denote this $n \times 1$ vector as $\mathbf{v}_i$. Similarly, $\mathcal{U}$ is the output word matrix. The $j-$th row of $\mathcal{U}$ is an $n-$dimensional embedded vector for word w_j when it is an output of the model. We denote this row of $\mathcal{U}$ as $\mathbf{u}_j$.

For this method sequence of actions can be written as follows:

- We generate our one hot word vectors $(\mathbf{x}^{(c-m)}, ..., \mathbf{x}^{(c-1)}, \mathbf{x}^{(c+1)}, ..., \mathbf{x}^{(c+m)})$ for the input context of size $2m$.
- We get our embedded word vectors for the context $(\mathbf{v}_{c-m} = \mathcal{V}\mathbf{x}^{(c-m)}, \mathbf{v}_{c-m+1} = \mathcal{V}\mathbf{x}_{(c-m+1)}, ..., \mathbf{v}_{c+m} = \mathcal{V}\mathbf{x}_{(c+m)})$
- Average these vectors to get $\tilde{\mathbf{v}} = \frac{\mathbf{v}_{c-m} + \mathbf{v}_{c-m+1} + ... + \mathbf{v}_{c+m}}{2m}$
- Generate a score vector $\mathbf{z} = \mathcal{U}\tilde{\mathbf{v}}$
- Turn the scores into probabilities
$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}) \tag{2.3}$$
- We desire our probabilities generated, $\hat{\mathbf{y}}$, to match the true probabilities, $\mathbf{y}$, which also happens to be the one hot vector of the actual word.

Described method can be interpreted as a feed forward neural network with one hidden layer that do not uses activation function. As a loss function for this algorithm can be chosen cross-entropy loss function

$$L = \sum_{i=1}^{|V|} \mathbf{y_i} \log(\hat{\mathbf{y}}_\mathbf{i}) \tag{2.4}$$

where $\hat{\mathbf{y}}$ is sofmatx (2.3) function.

2.1.2.2 Fast Text Word Embeddings

As mentioned previously, Fast Text method is a CBOW modification. Main modification is that Fast Text is taking into account not only words but also suffixes of words. The words are splitted into suffixes and as a result they can handle understanding of the context better.\

In this theses we used pretrained Fast Text models [5] consisting of 1 million word vectors trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens).

Chapter 3

Results Validation Theory

When the models are implemented and trained we have to compare them. This part is very important because we want to define such metrics that will not be biased and which will have high discriminability. In this project experiments are separated into two parts. First part is supervised classification with big amount of data. This is done for understating what is the maximal upper bound of specific classifiers. These upper bounds are used as maximums which our active learning algorithms must be converging to.

3.1 Receiving Operating Curve metric

In section 1 we mentioned that we are limiting our problem only on binary classification. Plenty of metrics such as recall, accuracy, precision, etc. exists for binary classification. However we decided to find a metic that is able to unify all metrics discussed before and do not underperform each of them. For these purposes we chose Receiving Operating Curve metric. ROC visualizes the tradeoff between true positive rate (TPR) and false positive rate (FPR). This means that for every threshold, we are able to calculate TPR and FPR and plot it on one figure.

We are working with balanced datasets. Thus, there is no problem in using ROC metrics. ROC metric is also very good when we care equally about positive and negative class. Another advantage is that if we notice small changes in ROC it will not result in big changes in other binary classification metrics.

Receiving Operating Curve lets us to calculate area under the curve (AUC). The probabilistic interpretation of ROC score is that if a positive case and a negative case are chosen randomly, the probability that the positive case outranks the negative case according to the classifier is given by the AUC.

3.2 Supervised Learning Results Validation

As mentioned above, supervised learning results are used as maximal upper bound of specific classifiers. In order to make results statistically valid we used k-fold cross validation. For each batch from k-fold cross validation we calculated both ROC and AUC. As an output result of a classifier performance we calculated mean value through all ROC and AUC results. All results are calculated with respect to balanced data classes.

3.3 Active Learning Results Validation

Active learning model evolution is based on supervised learning algorithms that are sequentially retrained. Thus, we are not able to display ROC for active learning algorithms because amount of results

is too big. We decided to aggregate results and display evolution of AUC metric for each step of active learning sequence. AUC sequences can be well compared between different classifiers. Another aspect of data validation is making the results statistically significant. We are not able to use k-fold cross validation for active learning algorithms. Therefore we active learning algorithm $H \in \mathbb{N}$ times. Due to the fact of random initializations, we are able to determine uncertainty bounds that are calculated as standard deviations to mean value.

Chapter 4

Data

Chapter 5

Implementation Architecture

Chapter 6

Passive Learning Classification as Upper Bound Threshold

Chapter 7

Active Learning Classification

Conclusion

Text of the conclusion...

Bibliography

[1] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.

[2] James O Berger. *Statistical decision theory and Bayesian analysis; 2nd ed.* Springer Series in Statistics. Springer, New York, 1985.

[3] Elizabeth D Liddy. Natural language processing. 2001.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[5] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[6] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[7] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural Computation*, 16(5):1063–1076, 2004.