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

Acknowledgment:
I would like to thank **doc. Ing. Václav Šmídl, Ph.D.** for his expert guidance and express my gratitude to Mgr. Yana Bilous for 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 Marko Sahan

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: Modely strojového účení pro klasifikaci jsou založené na účení parametrů black box modelu, které popisují vztáh mezi vzorky dat a jejích třídou. Proces sběru dat a jejích labelů pro účely trenování modelu muže být komplikováný a drahý. Množina dat je v mnoha případéch větši než množina dostupných labelů, ale našim pěedpokládem je to, že nové labely můžou byt obdržené prostřednictvím dotazu annotatorovi. Aktivní účení je proces výberu takových dat pro annotování, které přinesou největší diskriminabilitu o datasetu. Mnoho různých metodů aktivního účení v mnoha různých odvětvích bylo navřéno pro modely účení s učitélem. V tomto projektu jsou popsáné a ukázáné různé metody aktivního účení pro klassifikace textů. Navíc, jsou porovnavnáné už existující black box modely a jejích reprezentace neurčitosti. Modely aktivního účení jsou formalizováne pomoci teorie rozhodování, kde rozhodnutím je výběr dat bez labelů pro ziskávání annotace. Enropie predikce klassifikatoru je výbáaná jako očekaváná hodnota loss funkce rozhodování. Modely hlubokého účení ukázali state-of-the-art výsledky v ruzných odvětvích zpracování přirozeného jazyka a také i klassifikací textu. Kombinace aktivního účení pro výber dat a reprezentace neurcitosti, stavená na ensemblu hlubokých neuronových sití davá signifikantně lepší výsledky nez strategie náhodného výberu.

Klíčová slova: Aktivní účení, ensemble modely hlubokého účení, ensemble modely neuronoých sítí, klasifikace textu, teorie rozhodování, zpracování přirozeného jazyku.

Title:

Active learning for text classification

Author: Marko Sahan

Abstract: Machine learning approach to classification is based on learning parameters of black box model describing relation between the recorded data samples and their class labels. The process of data labels collection for the purposes of model training can be complex and costly. Therefore, the number of data record is often much higher than the number of labels, but the labels can be obtained by querying an annotator. Active learning is a process of selection of unlabeled data records for which knowledge of the label would bring the highest discriminability of the dataset. Various methods for active learning have been proposed in many different fields that use supervised learning models. In this project, we study suitability of various approaches for active learning of a text classification problem. We compare existing black box classifiers, and representations of their uncertainty. We formalize active learning using decision theory where the decision is which unlabeled data to select for annotation. The expected loss function of the decision making is chosen as entropy of the classifier predictions. Neural networks have showed state-of-the-art performance in different natural language processing tasks, including text classification.

The proposed combination of active learning data selection and uncertainty representation, based on deep learning ensembles algorithm achieves significantly better results than random data selection strategy.

Key words: Active learning, decision theory, deep learning ensembles, natural language processing, neural network ensembles, text classification.

Contents

Notation

Symbol	Definition
$\mathbf{x} \in \mathcal{X}$	vector of instance
$\mathbf{y} \in \mathcal{Y}$	one hot encoded label of a specific instance
$\mathbf{X} = \{\mathbf{x}_1, \ldots \mathbf{x}_M\}$	set of available instances
$\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_M\}$	set of labels that can be provided by an annotator
$\tilde{\mathbf{X}}$	set of training instances
$\tilde{\mathbf{Y}}$	set of training labels
$[x_1, x_2, ..., x_S]^T$	transposed vector notation
$(\mathbf{x}, \mathbf{y})$	tuple notation
$\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_R\}$	set notation
$a \in \mathcal{A}$	action from a set of all possible actions
$\theta \in \Theta$	decision theory uncertainty parameter
L	loss function
π^*	probability density function of variable θ
$p(\mathbf{y}\|\mathbf{x})$	probability density function of label **y** given instance **x**
$\hat{\mathbf{y}}$	estimate of **y** given **x**
$\mathbf{w}$	vector of SVM or decision tree weights
b	SVM bias (scalar)
$\mathbf{W}$	tuple of Naive Bayes, random forest or neural network (single layer) parameters
$\mathbf{b}$	vector of neural network single layer biases
Ω	tuple of neural network parameters (all weights and biases)
δ	Dirac delta function

Introduction

Active learning strategy lets the machine learning models iteratively and strategically query the labels of some instances for reducing human labeling efforts. This project shows how it is possible to connect active learning and text data. People has already been solving same problem for anomaly detection [6], image processing [8], etc.

If we take a look on modern approach of automating the labeling process of huge amount of unlabeled data, it is not optimal. People are randomly choosing unlabeled text data. These data are annotated by the subject matter experts, and used for training and testing the models. If the model performance is weak after the training, more text documents are selected and annotated. This approach is costly because nobody knows how much text documents must be selected to have good model scores. Our active learning strategy proposes selection of unlabeled text data that the model is not certain about. Unlabeled text data are given to a subject matter expert to provide the labels. Discussed problem was introduced almost two decades ago. Some active learning approaches for text classification dates back to 2001 [24], where are shown different querying strategies, and results of the active learning superiority over random sampling strategies. There is no clue that active learning strategy brings a lot of advantages. First of all, we are able to start with lower amount of training data, and iteratively extend the dataset. The dataset is extended using the data, which the model is not certain about. In this work we are extending our dataset with only one sample per active learning iteration. However, it was also shown that the strategies, which sample batches with more than one sample, also perform good results [2]. Thus, basing on the active learning approach, the model will get much more information from non-randomly chosen text samples.

The project describes different algorithms formulation with respect to decision theory, and then connection of all the methods to active learning theory. All methods used in this work are represented as ensemble models. Basing on [23] it was shown that ensemble deep learning algorithms give the best performance both for text and image processing data. Plenty of related active learning works for text classifications follow [8], with use of acquisition functions and dropout, for uncertainty representation e.g. named entity recognition [22] and text classification [13], [5]. We are concerned that ensembles outperform dropout uncertainty representation for text data, as it was shown in [11] for image classification.

This project also provides the link to Python implementation of active learning algorithms and comparison of different results gathered with respect to different data. We believe, that active learning approach is able to significantly reduce amount of time and expenses needed for automating text labeling process.

Chapter 1

Introduction to Decision Theory

The process of decision making is defined as a selection of the optimal action from a set of possibilities that can be applied at some operating conditions. The criteria of optimality is formalized by a loss function. The process of selection of the optimal action is, thus, formalized as the optimization problem. However, the operating conditions are often not known exactly, since we have incomplete information about them. Therefore, we will use the theory of decision making under uncertainty [3], where the uncertainty is represented by probability density functions. We will now briefly review the theoretical background.

1.1 Decision Theory

The theory of decision making has three basic elements: i) the set of possible actions $\mathcal{A}$ from which we should select an optimal action $a^* \in \mathcal{A}$, ii) the vector $\theta \in \Theta$ defining operating conditions under which we make the decision, where Θ is parameters space, iii) the loss function

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

that defines our preference of the action, in the sense that action a which has the lowest value from the action set of the loss function is preferred. For complete knowledge of the operating conditions θ the task is turned into simple optimization of (1.1). However, with incomplete information, we have to consider a range of possible states θ. The theory of decision making under uncertainty [3] defines the expected loss function that takes into account the uncertainty.

Definition 1. [3]If $\pi^*(\theta)$ is believed probability distribution of θ at the time of decision making, the *Bayesian expected loss* of an action a is

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

Based on definition 1.2, the optimal action is defined as the one that minimizes the expected loss:

$$a^* = \operatorname*{argmin}_{a \in \mathcal{A}} \mathbb{E}_{\pi^*}[L(\theta, a)]. \tag{1.3}$$

The key task of application of decision theory is the choice of the action space, parameter space, loss function, and method of evaluating the probability measure. In the following sections, we discuss examples of application of the theory to the problem of supervised learning and active learning, respectively.

1.2 Decision Theory for Supervised Learning

Supervised learning is defined as learning from the data with known target value. Specifically, for the classification problem, the target value is the class, where each data point belongs.

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 $\mathbf{x}$ is feature vector of size n, and $\mathbf{y}$ is its label assigned to the data instance $\mathbf{x}$ from space $\mathcal{X}$. Each value from space $\mathcal{Y}$ can be represented as a one hot representation, which is a vector consisting from ones and zeros. In the case of binary classification $\mathbf{y} \in \{[0,1]^T, [1,0]^T\}$, where the first class is represented as $\mathbf{y} = [1,0]^T$ and the 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 category, such as sports or comedy. As seen from this example, the label and the text are forming a tuple. In this work we are considering our data as tuples of variables $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$.

Basing on the data definitions from previous part, we can assume that $\mathcal{X} \times \mathcal{Y}$ is an infinite set and $(\mathbf{x}, \mathbf{y})$ is a sample from this set. We assume, that all available data tuples are sampled independently from a joint probability density function $p(\mathbf{x}, \mathbf{y})$. If $p(\mathbf{x}, \mathbf{y})$ was known, the optimal classifier $p(\mathbf{y}|\mathbf{x})$ can be obtained by the chain rule of probability

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x}). \tag{1.4}$$

However, since we do not know analytical form of the joint probability distribution, we aim at selecting the best possible approximation within a chosen class. Specifically, we choose a parametric form $p(\mathbf{y}|\mathbf{x}, a)$, where a is the parameter to be optimized.

The uncertainty of the decision task is a representation of the joint probability distribution. We will consider the uncertainty θ to be represented by empirical distribution:

$$\pi^* = p(\mathbf{x}, \mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N} \delta(\mathbf{x} - \mathbf{x}_i, \mathbf{y} - \mathbf{y}_i) \tag{1.5}$$

where $\mathbf{x}_i, \mathbf{y}_i$ are elements of training set ($\tilde{\mathbf{X}} \subset \mathcal{X}, \tilde{\mathbf{Y}} \subset \mathcal{Y}$). The training set is usually a subset of all available data on which the optimization is performed, the rest of the data is used for validation [25].

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, let us consider a linearly separable dataset. From the theoretical perspective, SVM constructs hyperplane in high dimensional space. 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 consider the uncertainty θ described with (1.5) that meets the condition of the limitation on $\Theta = (\tilde{\mathbf{X}} \subset \mathcal{X}, \tilde{\mathbf{Y}} \subset \mathcal{Y})$. Consider 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 the hinge loss function [20] 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 a 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 $\mathcal{Y}$. Thus, the uncertainty of the decision task can be defined as (1.5).

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

$$\rho(\mathbf{x}_i, \mathbf{w}, b) = \frac{1}{N}\sum_{i=1}^{N}\max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)). \tag{1.8}$$

1.2.2 Decision Theory and Algorithm Based on Neural Network Function

1.2.2.1 Neural Network

Given the instance $\mathbf{x} \in \mathcal{X}$ with further application of Feed Forward Neural Network (NN) we are able to predict an output value $\mathbf{y} \in \mathcal{Y}$. In this part of the work, our prior interest is focused around Feed Forward Neural Network algorithm, that assigns input value to a specific class.

First layer of NN is defined as

$$\mathbf{a}_1 = \mathbf{W}_1^T\mathbf{x} + \mathbf{b}_1, \tag{1.9}$$

where $\mathbf{W}_1$ is matrix of weights and $\mathbf{b}_1$ is a vector of bias values. First layer is called the input layer.

Further layers of NN are formed as

$$\mathbf{a}_k = \mathbf{W}_k^T f(\mathbf{a}_{k-1}) + \mathbf{b}_k,\ k = \{2, ..., K-1\}. \tag{1.10}$$

As seen from equation (1.10), neurons from each layer (except of input layer) take linear combination of the neurons from the previous layer. Function f is an activation function. Activation function is defined as a non-decreasing, continuous function. The most commonly used activation functions are sigmoid, relu, elu, and hyperbolic tangence functions [4].

Output values are computed with

$$\hat{\mathbf{y}} = f_{sm}\left(\mathbf{W}_K^T f(\mathbf{a}_{K-1}) + \mathbf{b}_K\right), \tag{1.11}$$

where f_{sm} is the softmax function, that is typically used for classification problems. The softmax function is defined as

$$f_{sm,i} = \frac{\exp(\mathbf{z}_i)}{\sum_{i=1}^{2}\exp(\mathbf{z}_i)},$$

where

$$\mathbf{z} = \mathbf{W}_K^T f(\mathbf{a}_{K-1}) + \mathbf{b}_K$$

is an output vector before activation function is applied. Output vector has the same size as label $\mathbf{y}$.

1.2.2.2 Decision Theory

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 $\hat{\mathbf{y}} = \hat{\mathbf{y}}(\mathbf{x}, \Omega)$ of the probability density function $p(\mathbf{y}|\mathbf{x})$, where $\mathbf{x}$ is the input data, $\Omega = (\mathbf{W}_1, ...\mathbf{W}_K, \mathbf{b}_1..., \mathbf{b}_K)$ is a collection of all neural network function parameters and 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

$$L(\mathbf{x}, \mathbf{y}, \Omega) = -y_1 \ln (\hat{y}_1(\mathbf{x}, \Omega)) - y_2 \ln ((\hat{y}_2(\mathbf{x}, \Omega)), \tag{1.12}$$

where $\mathbf{y} = [y_1, y_2]^T$ and $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2]^T$. With the usage of the given dataset, where $\forall i \in \{1, .., N\}$, $(\mathbf{x}_i, \mathbf{y}_i) \in (\tilde{\mathbf{X}} \subset \mathcal{X}, \tilde{\mathbf{Y}} \subset \mathcal{Y})$ are independent identically distributed, we can approximate $p(\mathbf{x}, \mathbf{y})$ as (1.5). Applying definition (1), expected loss for the algorithm based on a neural network function is evaluated as

$$\begin{aligned}
\mathbb{E}_{\pi^*} L &= \int_{\mathcal{X}\times\mathcal{Y}} L(\mathbf{x}, y, \Omega) p(\mathbf{x}, \mathbf{y}) d(\mathbf{x}, \mathbf{y}), \\
&= -\int_{\mathcal{X}\times\mathcal{Y}} \Big(y_1 \ln (\hat{y}_1(\mathbf{x}, \Omega)) + y_2 \ln ((\hat{y}_2(\mathbf{x}, \Omega))\Big) \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{x} - \mathbf{x}_i, \mathbf{y} - \mathbf{y}_i) d(\mathbf{x}, \mathbf{y}), \\
&= -\frac{1}{N} \sum_{i=1}^{N} \Big(y_1 \ln (\hat{y}_1(\mathbf{x}, \Omega)) + y_2 \ln ((\hat{y}_2(\mathbf{x}, \Omega))\Big),
\end{aligned} \tag{1.13}$$

where $\mathbf{y} = [y_1, y_2]^T$ and $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2]^T$.

1.2.2.3 Parameters Estimation

In further sections we are going to introduce more methods based on Neural Networks, which will slightly differ between each other. Thus, we would like to cover more theory around parameters estimation. The very simple, but efficient method is Gradient Descent. This method is based on equation

$$\hat{\Omega}_{n+1} = \hat{\Omega}_n - \eta_n \nabla L(\tilde{\mathbf{X}}, \tilde{\mathbf{Y}}, \hat{\Omega}_n, Z_n), \tag{1.14}$$

where $\hat{\mathbf{\Omega}}_n$ is the $n-$th iteration value of gradient descent of NN weights and its biases that converges to $\hat{\Omega}$. Value of $\nabla L(\mathbf{X}, \mathbf{Y}, \hat{\Omega}_n, Z_n)$ is a gradient of a loss function and η_n is $n-$th iteration of a value, that in terms of NN is defined as learning rate with a decay. Term Z_n represents indices from X and Y, that are used in $n-$th iteration of loss function. Learning rate decay is not obligatory feature. It can be constant as well. However, the best performance is obtained when the learning rate is estimated from the data, which is the main idea of adaptive methods, such as RMSProp or ADAM [10].

The usual gradient descent is an efficient method for complex $\hat{\mathbf{y}}(\mathbf{x}, \Omega)$ but may struggle with local minima. In this case, the algorithm may stop iterating even in a very shallow local minimum. Due to the complex functions $\hat{\mathbf{y}}(\mathbf{x}, \Omega)$, the loss function of its approximation will be non convex with large amount of local minima and maxima. Multiple solutions to this problem have been proposed in the form of Gradient Descent with momentum, or Stochastic Gradient Descent (SGD) [4]. In comparison to standard Gradient Descent, the key difference is that SGD allows us to use only one piece of data (minibatch) from the training dataset in order to calculate the step [4]. The minibatch is represented with value Z_n that says which training indices to use. The data samples (minibatch) are picked randomly at each step.

In this work, we are using the ADAM optimization [10], which is a stochastic gradient optimization, including adaptation of the learning rate as well as coefficient of the momentum.

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 a collection of parameters of individual classifiers, and will be used as an action ($a = \mathbf{W},\ a \in \mathcal{A}$) it the following decision task. 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 $p(\mathbf{y}|\mathbf{x}, \mathbf{W})$ normalization constant. The normalization constant would be dependent on the set of parameters $\mathbf{W}$. That fact would make our computations very complicated. In order to proceed with the loss function construction, we would like to go through Naive Bayes (NB) method.

1.2.3.1 Naive Bayes

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

$$p(\mathbf{W}|\mathbf{x}, \mathbf{y}) = \frac{p(\mathbf{y})p(\mathbf{x}|\mathbf{y}, \mathbf{W})p(\mathbf{W})}{\int_{\mathcal{W}} p(\mathbf{x}, \mathbf{y}|\mathbf{W})p(\mathbf{W})d\mathbf{W}}, \tag{1.15}$$

where $\mathcal{W}$ is the space of possible values of $\mathbf{W}$.

Naive Bayes method introduces very strong assumption in equation (1.15). 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 written as

$$\tilde{p}(\mathbf{W}|\mathbf{x}, \mathbf{y}) = \frac{1}{Z}p(\mathbf{y})p(\mathbf{W}) \prod_{i=1}^{n} (p(x_i|y_1, \mathbf{w}_i)^{y_1} p(x_i|y_2, \mathbf{w}_i)^{y_2}), \tag{1.16}$$

where $\mathbf{y} = [y_1, y_2]$, $\mathbf{W} = \{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_n\}$, and 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 (1.16) loss function L will be represented as

$$L(\mathbf{y}, \mathbf{x}, \mathbf{w}) = -\log(\tilde{p}(\mathbf{W}|\mathbf{x}, \mathbf{y}), \tag{1.17}$$

$$= \log(Z) - \log(p(\mathbf{y})) - \log(p(\mathbf{W})) - \sum_{i=1}^{n} \log(p(x_i|y_1, \mathbf{w}_i)^{y_1} p(x_i|y_2, \mathbf{w}_i)^{y_2}). \tag{1.18}$$

Same as in 1.2.1 and 1.2.2, we will assume that we can approximate $p(\mathbf{x}, \mathbf{y})$ as 1.5. From this moment, everything is ready for expected loss function derivation. Expected loss function for Naive Bayes method is derived as

$$\begin{aligned}
\mathbb{E}_{\pi^*}L &= \int_{\mathcal{X}\times\mathcal{Y}} L(\mathbf{x}, \mathbf{y}, \mathbf{w})p(\mathbf{x}, \mathbf{y})d(\mathbf{x}, \mathbf{y}),\\
&\int_{\mathcal{X}\times\mathcal{Y}} \Big(\xi(\mathbf{W}, \mathbf{y}) - \sum_{i=1}^{n} \log(p(x_i|y_1, \mathbf{w}_i)^{y_1} p(x_i|y_2, \mathbf{w}_i)^{y_2}\Big) \frac{1}{N}\sum_{j=1}^{N} \delta(\mathbf{x} - \mathbf{x}_j, \mathbf{y} - \mathbf{y}_j)d(\mathbf{x}, \mathbf{y}),\\
&\frac{1}{N}\sum_{j=1}^{N} \Big(\xi_j(\mathbf{W}, \mathbf{y}_j) - \sum_{i=1}^{n} \log(p(x_{i,j}|y_{1,j}, \mathbf{w}_i)^{y_{1,j}} p(x_{i,j}|y_{2,j}, \mathbf{w}_i)^{y_{2,j}})\Big),
\end{aligned} \tag{1.19}$$

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

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 a random forest theory.

1.2.4.1 Decision Tree

In this section, we expect our decision tree 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.), for each $\mathbf{x} \in \mathcal{X}$ 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 $\mathbf{w} \in \mathcal{W}$ exist a single upper bound, where $\mathcal{W}$ is redefined as a space of tree parameters. As a result, we will make all $\mathbf{w}$ to have the same length. If $\mathbf{w}$ has spare elements, they will be filled with zeros. Decision tree parameters space is $\mathcal{W}$. The loss function action $a \in \mathcal{A}$ will be represented as elements of the action space $\mathcal{A} = \mathcal{W}$. The loss function of a decision tree is a zero-one loss function, defined as

$$L(\mathbf{x}, \mathbf{y}, \mathbf{w}) = \begin{cases} 1, & \mathbf{y} \neq \hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}) \\ 0, & \mathbf{y} = \hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}) \end{cases}. \tag{1.20}$$

With the use 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 (1.5). As a result, the expected loss function for a 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 (1.20).

The conventional optimization procedure for decision trees is a heuristic, that works as follows. In the process of constructing a decision tree, we choose such feature $x_i \in [x_1, ..., x_n]^T = \mathbf{x}$ that will bring the highest information about the system (e.g. highest entropy of the features). This feature will form first layer on which a classification rule is designed. Then we add another feature with the highest information gain and construct the second layer. Using this method, we construct nodes and add more and more layers (branches). However, such a procedure can be sub-optimal, and improvement was achieved using a set of decision trees. For this purposes, we will define our decision tree $T(\mathbf{x}, \mathbf{w}_l)$, which has a one-hot encoded classification, i.e. two dimensional vector $\hat{\mathbf{y}}$ for the binary classification problem. Index l represents the set of parameters for the l-th three.

1.2.4.2 Random Forest

Consider $\mathbf{x} \in \mathcal{X}$ 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.

We will use the training set ($\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$), as a set from which we sample $V \in \mathbb{N}$ sets, $\tilde{\mathbf{X}}_v \subset \tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}_v \subset \tilde{\mathbf{Y}}$, $\forall v \in \{1, ..., V\}$. The data $\tilde{\mathbf{X}}_v$ and $\tilde{\mathbf{Y}}_v$ are created with random uniform sampling of indeces of instances in $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ without repetition. We, also, want each subset to contain strictly 70% of the data from $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$. As a result, the parameters space for random forests will form tuples of sets $(\tilde{\mathbf{X}}_v, \tilde{\mathbf{Y}}_v)$. Using this theory, we will construct V decision trees $\hat{\mathbf{y}}_v = T(\mathbf{x}, \mathbf{w}_v)$. As a result, the expected loss is

$$\mathbb{E}_{\pi^*} L = \frac{1}{V} \sum_{v=1}^{V} \frac{1}{N_v} \sum_{i=1}^{N_v} L(\mathbf{x}_{i,v}, \mathbf{y}_{i,v}, \mathbf{w}_v) \delta(\mathbf{x} - \mathbf{x}_{i,v}, \mathbf{y} - \mathbf{y}_{i,v}), \tag{1.21}$$

where $(\mathbf{x}_{i,v}, \mathbf{y}_{i,v}) \in (\tilde{\mathbf{X}}_v, \tilde{\mathbf{Y}}_v)$ and N_v is the number of the data samples in $\tilde{\mathbf{X}}_v$ and $\tilde{\mathbf{Y}}_v$. If we assume $\mathbf{w}_v$ to be a random variable, then V decision trees form samples from the probability density function $p(\mathbf{y}|\mathbf{x}, \mathbf{w})$. In other words

$$p(\mathbf{y}|\mathbf{x}, \mathbf{w}_v) = \hat{y}_{v,1}^{y_1} \hat{y}_{v,2}^{y_2}, \tag{1.22}$$

where label $\mathbf{y}$ is written as a one-hot representation, and the estimate is in the same form $\hat{\mathbf{y}}_v = [\hat{y}_{v,1}, \hat{y}_{v,2}]^T$. Thus, we can say that classification probability $p(\mathbf{y}|\mathbf{x})$ can be written as

$$p(\mathbf{y}|\mathbf{x}) = \int_{\mathbf{w} \in \mathcal{A}} p(\mathbf{y}|\mathbf{x}, \mathbf{w}) p(\mathbf{w}) d\mathbf{w}, \tag{1.23}$$

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

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{V} \sum_{l=1}^{V} \hat{y}_{v,1}^{y_1} \hat{y}_{v,2}^{y_2}, \tag{1.24}$$

where each decision tree $T(\mathbf{x}, \mathbf{w}_v)$ is constructed with the use of (1.21).

In order to proceed with further sections we define the output of Random Forest as $\hat{\mathbf{y}} = \hat{\mathbf{y}}(\mathbf{x}, \mathbf{W})$, where $\mathbf{W} = \{\mathbf{w}_1, ..., \mathbf{w}_V\} \in \mathcal{W}$ is a set of parameters of specific Random Forest algorithm. We define vector $\hat{\mathbf{y}}$ as

$$\hat{\mathbf{y}} = \frac{1}{V} \sum_{l=1}^{V} \hat{\mathbf{y}}_v. \tag{1.25}$$

1.3 Decision Theory for Active Learning

As mentioned in previous sections, the training set $\tilde{\mathbf{X}} \times \tilde{\mathbf{Y}}$ is only a subset of all available data. It is important that each point $\mathbf{x}$ in the training set has a label $\mathbf{y}$. In many situations labels are expensive to obtain, and the number N of all available samples $\mathbf{x} \in \mathbf{X}$ is large, while labels are available only for the initial subset $J_0 = \{1, \dots, N_0\}, N_0 \ll N$. This problem is known as semi-supervised learning.

We consider a setup in which we can ask for a label for an arbitrary $\mathbf{x}$, that can be provided for example by a human (annotator). We assume that getting labels needs some time and is very expensive. The task is to choose which sample we will ask to label.

The active learning problem is defined as a sequence of supervised learning problems. Specifically, we assume that the initial sets for supervised learning are $\mathbf{X}_0 = \{\mathbf{x}_i\}_{i \in J_0}$ and $\mathbf{Y}_0 = \{\mathbf{y}_i\}_{i \in J_0}$. 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 the 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 as a space of the data indices of the unlabeled data $\mathbf{x}$, $a = j_u \in \mathcal{A}_u = J \backslash J_u$. The uncertainty of the decision task, if the parameter space Θ of the used classifiers, i.e. $\theta = (\mathbf{w}, \mathbf{b})$ for SVM, $\theta = \Omega$ for NN, $\theta = \mathbf{W}$ for RF and NB. It is very vital to understand, that point estimate of the parameters that was designed in supervised learning is not sufficient for this task. We need full distribution on the parameters because we will integrate over the parameters space. As an example, if we talk about the 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 the same.

The decision task for this particular problem can be written as

$$j_u^* = \underset{j_u \in J \setminus J_u}{\operatorname{argmin}}(\mathbb{E}_{\pi_u^*} L^*), \tag{1.26}$$

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

$$\mathbb{E}_{\pi_u^*} L^* = \int_{\Theta} L^*(j_u, \theta) \pi_u^* d\theta, \tag{1.27}$$

where $j_u \in J \setminus J_u$, $\theta \in \Theta$ 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.

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

Consider 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 a 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 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 (1.27). Mentioned distribution is a distribution of the models' parameters, given the training data that can be written as

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

We do not have explicit form of the pdf. However, we assume that we have 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

$$\pi_u^* = \frac{1}{Q_u} \sum_{q=1}^{Q_u} \delta(\theta_u - \theta_u^{(q)}), \tag{1.29}$$

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, it is 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 $\tilde{\mathbf{Y}}$. Let us rewrite it in more mathematical form.

Assume N_u is amount of samples in $\mathbf{X}_u$. Let $Z_u = \{z_1, ..., z_{N_u^{sub}}\} \subset J_u$, where $N_u^{sub} < N_u$. Let $p(\theta_u|\mathbf{X}_u, \mathbf{Y}_u, Z_u)$ be a probability of model parameters θ_u given $\mathbf{X}_u, \mathbf{Y}_u$ and Z_u. The conditioning in the pdf is defined as restriction of sets $\mathbf{X}_u$, $\mathbf{Y}_u$ on indices from Z_u. As a result we can approximate (1.28) as

$$\begin{aligned}
\pi_u^* &= p(\theta_u|\mathbf{X}_u, \mathbf{Y}_u) && (1.30)\\
&= \int p(\theta_u|\mathbf{X}_u, \mathbf{Y}_u, Z_u)p(Z_u)dZ_u && (1.31)\\
&= \frac{1}{Q_u}\sum_{q=1}^{Q_u} p(\theta_u|\mathbf{X}_u, \mathbf{Y}_u, Z_u^{(q)}) && (1.32)\\
&= \frac{1}{Q_u}\sum_{q=1}^{Q_u} \delta(\theta_u - \theta_u^{(q)}). && (1.33)
\end{aligned}$$

Sampling from $p(Z_u)$ is very simple. The only thing that must be predefined is N_u^{sub}. After training the model using $\mathbf{X}_u$ and Y_u under restriction 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 network based classifiers. SGLD modifies neural network learning algorithm by adding noise in Stochastic Gradient Descent. In comparison to normal Feed Forward Neural Network in [27] is introduced Stochastic Gradient Langevin Dynamics Neural Network (SGLD). This is a type of Bayesian Neural Network algorithm. In comparison to normal Feed Forward NN, [27] proposes to add Gaussian noise to the system while doing Gradient Descent. In fact, when the algorithm is almost trained, additional noise let us to sample i.i.d parameters values in a neighborhood of a minimum. With the usage of Bayes rule we can rewrite (1.28) as

$$\begin{aligned}
\pi_u^* &= p(\theta_u|\mathbf{X}_u, \mathbf{Y}_u)\\
&\propto p(\mathbf{X}_u, \mathbf{Y}_u|\theta_u)p(\theta_u). && (1.34)
\end{aligned}$$

Next, we can approximate (1.34) as

$$\pi_u^* = \frac{1}{Q_u}\sum_{q=1}^{Q_u} p(\mathbf{X}_u, \mathbf{Y}_u|\theta_u^{(q)}) \qquad (1.35)$$

that results in (1.33). However, for this case θ must meet some constraints. If we want to sample parameters from its distribution, θ must be independent identically distributed. Basing on [27] the updated gradient for SGLD can be written as

$$\begin{aligned}
\hat{\Omega}_{n+1} &= \hat{\Omega}_n - \frac{N}{N_{minibatch}}\frac{\eta_n}{2}\Big(\nabla L(\tilde{\mathbf{X}}, \tilde{\mathbf{Y}}, \hat{\Omega}_n, Z_n)\Big) + \epsilon_n, && (1.36)\\
\epsilon_n &\sim \mathcal{N}(0, \eta_n I), && (1.37)
\end{aligned}$$

where N is an amount of training data, $N_{minibatch}$ is an amount of the data in minibatch and η_n is $n-$th iteration of a learning rate.

1.3.1.3 Dropout

Dropout is another technique similar to SGLD how to sample from a neural network parameters distribution. This type of sampling can be also considered as a Bayesian Neural Network [8]. The idea is randomly turn off some neurons while training. Thus, an estimate of $\pi_u^* = p(\theta_u|\mathbf{X}_u, \mathbf{Y}_u)$ is the Dirac function as well 1.29. When the algorithm is almost trained, we are able to start sampling the i.i.d. parameter masks. The process of generating samples $\theta^{(q)}$ are generated after each step of the SGD, where different mask of neural weight and biases is sampled for each q.

1.3.1.4 DENFI

DENFI algorithm is an extension of deep ensembles [11]. In case of DENFI algorithm we can approximate (1.28) with the usage of equations (1.34) and (1.33). The equations are same but sampling from $p(\theta_u)$ is different. In this work we are using modification of the initial DENFI algorithm. The idea of this algorithm is to train ensemble of Q_u Feed Forward Neural Networks with the usage of Stochastic Gradient Descent. In theory, each neural network will find different local minimum due to different initial weights of neurons and Stochastic Gradient Descent. The beauty of this algorithm is in further training iterations that will be described and shown in further sections. For now, we are only interested in parameters sampling from $p(\theta_u)$, that has been already covered and described.

1.3.2 Active Learning Loss Function

The acquisition function is a function that helps us to decide which data sample is the best for model learning given model's uncertainty. Wide overview of different acquisition functions is shown in [8] e.g. entropy, information or mean standard deviation maximization. However, in our work we decided to use only the entropy loss.

1.3.2.1 Entropy Based Active Learning Loss

First approach of defining Active Learning loss function is negative entropy. Basing on the formal entropy definition [21], we can write it as

$$-H(\hat{\mathbf{y}}|\mathbf{x}_{j_u}, \theta_u) = \sum_{r=1}^{R} \hat{y}_r(\mathbf{x}_{j_u}, \theta_u) \log\left(\hat{y}_r(\mathbf{x}_{j_u}, \theta_u)\right), \tag{1.38}$$

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

$$\begin{aligned}
\mathbb{E}_{\pi_u^*} L^* &= \int_{\Theta_u} -H(\hat{\mathbf{y}}|\mathbf{x}_{j_u}, \theta_u) p_u(\theta_u|\mathbf{X}_u, \mathbf{Y}_u) d\theta_u \\
&= \int_{\Theta_u} -H(\hat{\mathbf{y}}|\mathbf{x}_{j_u}, \theta_u) \frac{1}{Q_u} \sum_{q=1}^{Q_u} \delta(\theta_u - \theta_{u,q}) d\theta_u \\
&= \frac{1}{Q_u} \sum_{q=1}^{Q_u} -H(\hat{\mathbf{y}}|\mathbf{x}_{j_u}, \theta_{u,q})) \\
&= \frac{1}{Q_u} \sum_{q=1}^{Q_u} \sum_{r=1}^{R} \hat{y}_r(\mathbf{x}_{j_u}, \theta_{u,q}) \log\left(\hat{y}_r(\mathbf{x}_{j_u}, \theta_{u,q})\right).
\end{aligned} \tag{1.39}$$

As a result, minimization of given expected loss will lead us to a sample with the highest entropy. Thus, we are seeking for a maximal entropy data sample index. In other words $j_u^* = \operatorname{argmin}_{j \in J \setminus J_u}(\mathbb{E}_{\pi_u^*} L^*)$.

1.3.3 Active Learning

We would like to generalize active learning part for all described algorithms, in order to estimates $p(\mathbf{y}|\mathbf{x}, \mathbf{X}_u, \mathbf{Y}_u)$ basing on samples from π_u^* in (1.28). In Supervised Learning section we have derived estimate $\hat{\mathbf{y}}$ of $\mathbf{y}$ for SVMs, Random Forests and Feed Forward Neural Networks. Active Learning algorithm requires distribution over the parameters of the algorithms. We will solve this problem the way that we will get samples from π_u^*, and then approximate probability distribution as (1.28).

In order to estimate $p(\mathbf{y}|\mathbf{x}, \mathbf{X}_u, \mathbf{Y}_u)$, we define Generalized Ensembles Algorithm. SGLD and DENFI can be also represented as generalized ensembles models because parameters sampling (neuron weights sampling) represents different configurations of neural networks. Thus, each sample from π_u^* can be assumed as i.i.d. ensemble. As a result, we will use Q_u ensembles in each step of Active Learning algorithm. Therefore, basing on the previous theory we can approximate $p(\mathbf{y}|\mathbf{x}, \mathbf{X}_u, \mathbf{Y}_u)$ as

$$p(\mathbf{y}|\mathbf{x}, \mathbf{X}_u, \mathbf{Y}_u) = \frac{1}{Q_u} \sum_{q=1}^{Q_u} \hat{\mathbf{y}}_{q,u}. \tag{1.40}$$

1.4 Conclusion

In this section, we have covered the decision theory for both passive and active learning with respect to different algorithms and ensemble approaches. Passive Learning section showed how it is possible to represent SVM, Random Forest, and neural networks in terms of decision theory problem setup. In addition to this, Active Learning section showed how to represent the uncertainty of the model and parameters sampling for ensembles representation.

Chapter 2

Natural Language Processing Theory

2.1 Text Representation

According to [12], 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 [19] and Fast Text Word Embeddings [15]. 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 is used in the thesis is the 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 efficient. 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 of two parts as 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 assume words from corpus D. Thus, the denominator is always greater than zero.

Finally,

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

2.1.1.1 TF-IDF and Information Theory

In this part is shown the connection of TF-IDF to information theory [1]. Let us 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 the vectors that represent each document, will be.

2.1.2 Fast Text and CBOW Word Embeddings

Term "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. Plenty of word embedding methods based on neural networks and co-occurrence matrices exist nowadays. 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

The CBOW embeddings have been introduced in [14]. Consider the sentence "A beautiful cat jumped over a puddle". We choose the window of $2m+1,\ m \in \mathbb{N}$ words and try to predict the word with index

$m + 1$, basing on the context of size $2m$, where m words are before the prediction word and m words are after the prediction word. We would like to treat tuple ("a", ”beautiful”, "cat", ’over", "a’, "puddle") as a context, and based on the context we would like to predict or generate the word "jumped". This type of model is the Continuous Bag of Words (CBOW) model. Let the known parameters in our model be the contexts, represented with one-hot encoded word vectors. The input one hot encoded word vector is denoted as $\mathbf{c}^{(c)}$, where c is the position of the word in the context. The predicted context word is defined as $\mathbf{c}_{\text{pred}}$. We create two matrices, $\mathbf{E} \in \mathbb{R}^{E\times|\mathbf{v}|}$ and $\mathbf{U} \in \mathbb{R}^{|\mathbf{v}|\times E}$. Where E is an arbitrary size which defines the size of our embedding space and $|\mathbf{v}|$ is a vocabulary size of all words from all contexts. $\mathbf{E}$ is the input word matrix, such that the i-th column of $\mathbf{E}$ is the E-dimensional embedded vector for word $\mathbf{c}_i$. We denote this $E \times 1$ vector as $\mathbf{e}_i$. Similarly, $\mathbf{U}$ is the output word matrix. The $k-$th row of $\mathbf{U}$ is an E-dimensional embedded vector for word $\mathbf{c}_{\text{pred},k}$ when it is an output of the model. We denote this row of $\mathbf{U}$ as $\mathbf{u}_k$.

The matrices $\mathbf{E}$ and $\mathbf{U}$ are obtained by the following sequence of actions:

- We generate our one hot word vectors $(\mathbf{c}^{(c-m)}, ..., \mathbf{c}^{(c)}, ..., \mathbf{c}^{(c+m)})$ for the input context of size $2m$
- We get our embedded word vectors for the context $(\mathbf{e}_{c-m} = \mathbf{E}\mathbf{c}^{(c-m)}, \mathbf{e}_{c-m+1} = \mathbf{E}\mathbf{c}^{(c-m+1)}, ..., \mathbf{e}_{c+m} = \mathbf{E}\mathbf{c}^{(c+m)})$
- Average these vectors to get $\tilde{\mathbf{e}} = \frac{\mathbf{e}_{c-m}+\mathbf{e}_{c-m+1}+...+\mathbf{e}_{c+m}}{2m}$
- Generate a score vector $\mathbf{z} = \mathbf{U}\tilde{\mathbf{e}}$
- Turn the scores into probabilities
$$\hat{\mathbf{c}}_{\text{pred}} = \text{softmax}(\mathbf{z}) \tag{2.3}$$
- We optimize $\mathbf{E}$ and $\mathbf{U}$ to maximize the match between the predicted vector $\hat{\mathbf{c}}_{\text{pred}}$, and the true vector $\mathbf{c}_{\text{pred}}$, 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 only input and output layers. The optimization is provided with respect to cross entropy loss function

$$L = \sum_{i=1}^{|\mathbf{v}|} \mathbf{c}_{\text{pred},i} \log(\hat{\mathbf{c}}_{\text{pred},i}), \tag{2.4}$$

where $\hat{\mathbf{c}}_{\text{pred}}$ is softmax (2.3) function. When the weights in neural network are trained, matrix $\mathbf{E}$ represents $|\mathbf{v}|$ word embeddings where i-th word from vocabulary $\mathbf{v}$ is i-th row in $\mathbf{E}$.

Previously, we mentioned that all the methods take an instance (text) $\mathbf{x}$ as an input value and predict its class. Instance $\mathbf{x}$ is calculated as a mean value from all words embeddings in the text

$$\mathbf{x}_i = \frac{1}{|\mathcal{D}_i|} \sum_{j\in\mathcal{D}_i} \mathbf{E}\mathbf{c}^{(j)},$$

where $\mathcal{D}_i$ is the set of indices of all words in i-th document in the common vocabulary.

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 thesis we used pretrained Fast Text models [15] consisting of 1 million word vectors trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens).

Chapter 3

Data and Evaluation Metrics

3.1 Evaluation Metrics

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 a supervised classification with big amount of the training data. This is done for classifiers' maximal upper bound understating. The upper bounds are used as maxima to which our active learning algorithms should be converging.

3.1.1 Receiver Operating Characteristic 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 scoring. However, we decided to find a metric that is able to unify all metrics discussed before and do not underperform each of them. For these purposes we chose Receiver Operating Characteristic (ROC) metric. ROC visualizes the tradeoff between true positive rate (TPR)

$$\text{TPR} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

and false positive rate (FPR)

$$\text{FPR} = \frac{\text{false positive}}{\text{false positive} + \text{true negative}},$$

where terminology true/false positive/negative refers to assigned classification being correct or incorrect with respect to positive or negative category [7]. This means that for every threshold we are able to calculate TPR and FPR, and plot it in one figure.

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

The metric results should be aggregated into a single number in order to obtain a unique comparison value. This is typically done as calculation of the area under the ROC, which is known as the area under the curve (AUC) metric. The probabilistic interpretation of ROC score means 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 [7].

3.1.2 Supervised Learning Results Validation

As mentioned above, supervised learning results are used as a maximal upper bound of the specific classifier. 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 over all ROC and AUC scores. All results are calculated with respect to balanced data classes.

3.1.3 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 the amount of results is too big. We decided to aggregate results and display evolution of AUC metric for each step of the active learning sequence. AUC sequences can be well compared over 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 run the active learning algorithm $H \in \mathbb{N}$ times with different random selection of the initial training set. Due to the random initializations, we are able to determine uncertainty bounds that are calculated as standard deviations from the mean values.

3.2 Data

This chapter is dedicated to dataset description. We used two datasets for algorithms training and testing. We consider these datasets big and diverse enough for getting unbiased results. We took into account size of texts (articles and tweets), diversity and at the same time, similarity of topics.

3.2.1 HuffPost 200k Articles Dataset

HuffPost 200k Articles Dataset is publicly available at Kaggle competition webpage and can be found as News Category Dataset [16] here `https://www.kaggle.com/rmisra/news-category-dataset`. Described dataset contains around 200k news headlines from the year 2012 to 2018 obtained from HuffPost web journal. Following dataset includes url address and label to each article. HuffPost dataset has 200k articles assigned to 41 categories. We used only 10 categories. We are interested in binary classification, as a result we have to make pairs from chosen categories. These categories and pairs are listed in table 3.1.

Tuple Id	Category Pairs	
1	Crime	Good News
2	Sports	Comedy
3	Politics	Business
4	Science	Tech
5	Education	College

Table 3.1: HuffPost Dataset Categories which were chosen for algorithms' training and testing

Due to the fact that there is no raw article included in the dataset, we used url links in order to find the articles. For each category we scraped 500 original articles from `www.huffpost.com`. Listed categories are chosen with respect to diversity and classification complexity. We sorted the categories in table 3.1 with respect to ascending classification complexity order. By classification complexity we mean two sets intersection in feature space. If the classification complexity is high, majority of feature space dimensions have intersections between two datasets. Thus, it is harder to find such set of features that can be used for high classification performance.

3.2.2 1600k Tweets Dataset

Another dataset that is used in this work is 1600k tweets dataset, which is publicly available and is also taken from Kaggle competition webpage. The dataset can be found as sentiment140 dataset [9] at `https://www.kaggle.com/kazanova/sentiment140`. This dataset contains 1,600,000 tweets extracted using Twitter API. The tweets have been annotated as negative (0), positive (4) and they are used for sentiment detection. Same as with HuffPost dataset we used 500 records for each category, for training and testing purposes. The reason of choosing this dataset is that we wanted to show how our algorithms can handle data that consist of little texts.

Chapter 4

Project Implementation and Architecture

Even though this work is quite theoretical with experiments that proof theoretical concepts, we consider implementational part interesting as well. In this chapter we show the architecture of the project and explain how different dependencies cooperate with each other. The codebase of this project can be easily found at `https://github.com/sahanmar/Peony`. We expect this project to grow continuously and be used not only in terms of master thesis.

This project was written in Python 3.7 programming language with the usage of Conda environment. The project combines a lot of different tools and programs such as Docker, Docker-Compose, MongoDb, Jupyter, etc..

In this thesis we used two main components that represent the database and a computational part. We tried to unify all methods as much as possible and make the utilization process very easy.

4.1 Database

In order to make everything consistent and let the models work with the same input and output format we decided to create a database that will store all the data in JSON format. This unification let us connect the database to machine learning and visualization components. In this project we decided to work with NoSQL database. Our choice was MongoDb. The reason why we have chosen MongoDb is because of its simplicity and possibility of maintaining through Docker. Since Docker and MongoDb is a perfect combination, the database can be deployed with two lines of code through Docker-Compose as explained in documentation on GitHub. Of course it is easier to use MongoDb without Docker but our motivation was measured on simplicity of creating and working with the database. All experiments were run on Google Cloud Platform virtual machine instance. Thus, we could start working with the models right away without any complications with installation.

4.1.1 MongoDb Data Format

MongoDb represents the data in BSON format behind the scenes but we insert and get the data in JSON format. Despite the fact that we are having different text datasets which we store in the database, we decided to create a unified JSON scheme that will let us preserve the structure of the data stored in MongoDb. JSON schema of how the data are stored and what a user will get as an output from a database is shown in figure 4.1. Deeper explanation of JSON schema format can be found at `https://json-schema.org/understanding-json-schema/`.

```
{
  "title": "Database",
  "type": "object",
  "properties": {
    "datasetName": {
      "type": "string",
      "description": "Name of the dataset"
    },
    "datasetId": {
      "type": "int",
      "description": "Unique hash id that will be created automatically"
    },
    "record": {
      "type": "object",
      "description": "All information about an instance",
      "properties": {
        "id": {
          "type": "string",
          "description": "Unique hash id that will be created automatically
              "
        },
        "snippet": {
              "type": "string",
              "description": "Snippet of a text. Can be empty"
        },
        "text": {
          "type": "object",
          "description": "Text instance that is used for a model",
          "properties" : {
            "title": {
              "type": "string",
              "description": "Title of a text. Can be empty"
            },
            "body": {
              "type": "string",
              "description": "Body of a text"
            },
          },
        },
        "label": {
          "type": "string",
          "description": "Label for an instance. Can be empty if this is
              not validation data"
        },
        "metadata": {
          "type": "object",
          "description": "Any additional metadata. Can be empty field"
        },
      },
    },
  },
}
```

Figure 4.1: MongoDb JSON schema visualization

4.2 Computations

All computations were done with the usage of virtual instance on Google Cloud Platform. We used configuration with 2 CPU, 7.5Gb memory that was running on Debian GNU/Linux 10. All versions of python, python packages, docker, mongo, etc. can be found in .yml files in GitHub folder with the project.

4.3 Machine Learning Component

Machine Learning (ML) Component is fully implemented in Python with the usage of open source libraries. In order to understand how to use the models, it is possible to find the code and its usage in Jupyter notebook that are stored in showcase folder. Showcase folder has four Jupyter notebooks that show both how to get the data for the models from the database and how to start using the models.

4.3.1 Data Transformers

Before models training and testing, user has to fit the data transformer that transforms text into tensors form. Tensors are used as input values for models. As mentioned in chapter 2, we are working only with Fast Text and TF-IDF text encodings. Both TF-IDF and Fast Text models are fitted from the documents that are given to the transformer.

4.3.1.1 TF-IDF Transformer

Basic concept is that TF-IDF transformer represents one article as a vector. TF-IDF encoding for a single document is calculated on the basis of all extracted words that exist in the vocabulary. As a result, if we make TF-IDF encoding of a set of articles, we will get a matrix where each row represents specific document and each column represents word from a dictionary.

4.3.1.2 Fast Text Transformer

Fast Text model is a pretrained model that consists of one million words mapped to vectors. These words are stored in MongoDb. When a user starts to use Fast Text model, ML component creates a words' vocabulary from the texts taken for model training/testing. This vocabulary is created in a form of a hash map (word -> vector) where word embeddings are downloaded from MongoDb. It is important to remember that Fast Text encoding represents each word as a vector with predefined number of components. We are using word embeddings that represent each word with 300 float values. We introduce article encoding as a mean value through all words from a text that is given for encoding. Thus, if we make Fast Text encoding of a set of articles, we will get a matrix where each row represents specific document. Huge advantage of this method in comparison to TF-IDF is that we are working only with 300 float values than with huge vocabulary (all unique words from a text corpus). Therefore, we get both lower features dimensionality and better context understanding.

4.3.2 Machine Learning

In this work we created a Generalized Model that unifies all models. Generalized Model allows us to work with each Machine Learning algorithm in the same way. Generalized model is able to take a data transformer as an input argument. This feature makes it easier to work with models. In first chapter of this thesis we introduced our models the way that we want to sample from their parameters' distributions.

In other words we defined ensembles models. In figure 4.2 is shown a generalized diagram of machine learning structure.

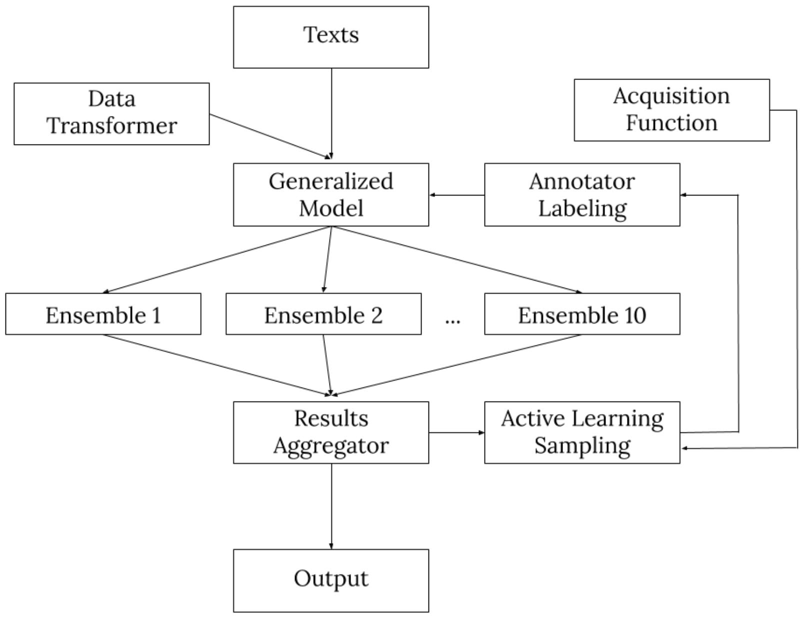

Figure 4.2: Machine Learning Workflow

We used scikit-learn [18] for basic algorithms such as Random Forests and SVMs. However, core of this project is constructed around neural networks. We used PyTorch [17] as a neural networks framework. In this work we have implemented and tested five classification algorithms. Three of them, such as SVM, Random Forest and Feed Forward Neural Network ensembles are trained on a randomly chosen subsets from the training data. For each ensemble are randomly chosen 80% from training data that are used for training. Two algorithms such as SGLD and DENFI are using full training dataset for their ensembles. The variability in SGLD and DENFI ensembles is reached through adding Gaussian noise while ensembles training. We hardcoded amount of ensembles for all models to 10, except SGLD where are used 50 ensembles.

4.3.2.1 SVM and Random Forest Ensemble Setup

Both SVM and Random Forest models were taken and used out of the box. We created 10 SVM and 10 Random Forest ensembles with default scikit-learn setting. No modifications were provided.

4.3.2.2 Feed Forward Neural Network

Despite the fact that we used very simple neural networks architecture it showed great results. We implemented Feed Forward Neural Network with the usage of PyTorch python package with only input layer that consists of 100 neurons with sigmoid activation function. We chose softmax activation function for output layer.

Algoritmus 4.1 DENFI modification algorithm pseudocode

```
def active_learning_iteration_training(all_ensembles, training_data):
  for ensemble in all_ensembles:
    if first_active_learning_iteration is False:
      ensemble.weights = ensemble.weights_prev_iteration
      training_epochs = 500
    else:
      ensemble.weights = gaussian_initialization(mean=0, var=0.1)
      training_epochs = 2000
    ensemble.train(training_data)
        ensemble.weights = ensemble_weights
                         + gaussian_noise(mean=0, var=0.1)
```

4.3.2.3 SGLD

For SGLD we used the same configuration as for Feed Forward Neural Network. One significant difference is that we used whole training dataset and were adding Gaussian noise to a Gradient Descent [27] as shown in (1.36). Precise configuration of SGLD can be found in github project here `https://github.com/sahanmar/Peony/blob/master/Peony_project/Peony_box/src/peony_adjusted_models/sgld_nn.py`.

4.3.2.4 DENFI

In this work we have simplified original DENFI algorithm. The pseudocode of this algorithm is shown in algorithm 4.1. The main idea is that the algorithm finds different local minimums due to the random weights initialization in first active learning iteration. When the training is finished, Gaussian noise is added to the output weights in order to increase the variability. In further active learning iterations, weights from previous iterations with extended training dataset is used. After the training we also add Gaussian noise to the weights. When the learning iterations are done, we can use the algorithm for predicting.

Chapter 5

Passive Learning Classification

Before we can start active learning part we have to understand if implemented algorithms are capable to solve the classification task. Thus, we decided to test the models on Sports and Comedy categories from HuffPost Dataset and on tweets from Tweets Dataset. The classification was done both for TF-IDF and Fast Text encodings. We separated experiments with respect to the dataset types.

5.1 Passive Learning HuffPost Dataset

In this section we are illustrating ROC and AUC metrics with respect to $10-$fold cross validation and 500 Sports, 500 Comedy articles. Vocabulary, that is created from 1000 articles corpus consists of 20 thousand unique words. We would like to mention that all algorithms are trained and tested with respect to 10 ensembles. Moreover, the ratio of randomly chosen training data for ensembles (SMV, Random Forest, Feed Forward NN ensembles) is set to 70%.

5.1.1 SVM Ensembles

Results for SVM Ensembles shown in figures 5.1 and 5.2. These are results both for TF-IDF and Fast Text encodings. As seen on these plots ROC and AUC values of $10-$fold cross validation are extremely high. This means that our model works very good. Another interesting point is that standard deviation with respect to all runs is very low. It means that SVM ensembles could linearly separate Sports and Comedy sets with acceptable classification error.

It is also seen that ROC and AUC metrics are almost same for TF-IDF and Fast Text encodings. However, there is one significant difference. The difference is in computational time because Fast Text encoded text document consists of 300 float components and TF-IDF encoded text document consists of 20 thousand float components. Even though we are using algorithms for sparse matrix computations for TF-IDF method, computations for Fast Text encoding are approximately five times faster. The higher amount of unique words is, the higher elapsed time per article for TF-IDF based encoding algorithm will be. Another good aspect is algorithm's simplicity. SVM is trained much faster than neural network based models.

It is possible to conclude that the model shows good results for this classification problem and can be used for active learning experiments.

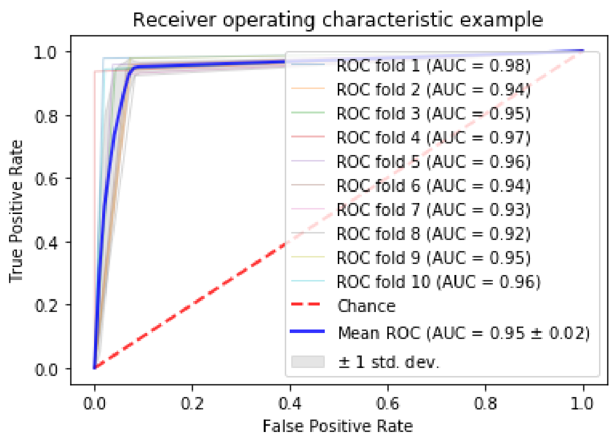

Figure 5.1: TF-IDF ROC and AUC for 10 SVM ensembles trained and tested on Sports and Comedy data where each ensemble is trained on 70% of randomly chosen training data

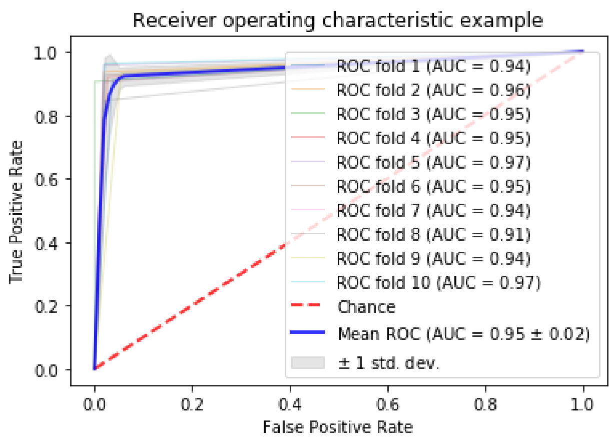

Figure 5.2: Fast Text ROC and AUC for 10 SVM ensembles trained and tested on Sports and Comedy data where each ensemble is trained on 70% of randomly chosen training data

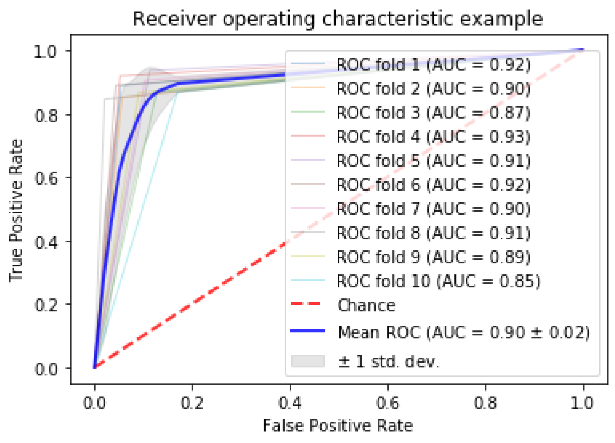

Figure 5.3: TF-IDF ROC and AUC for 10 Random Forest ensembles trained and tested on Sports and Comedy data where each ensemble is trained on 70% of randomly chosen training data

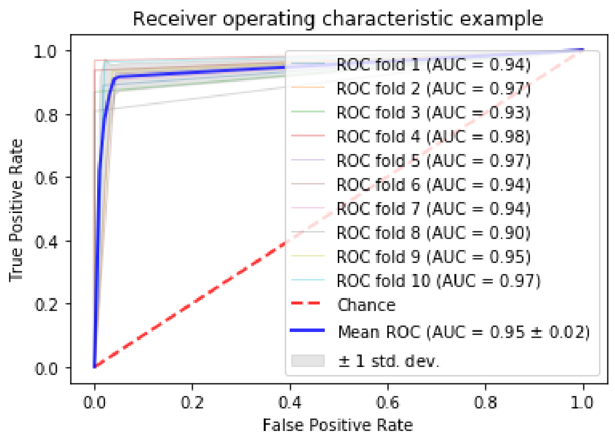

Figure 5.4: Fast Text ROC and AUC for 10 Random Forest ensembles trained and tested on Sports and Comedy data where each ensemble is trained on 70% of randomly chosen training data

5.1.2 Random Forest Ensembles

Results for Random Forest ensembles are shown in figures 5.3 and 5.4. Illustrated results are both for TF-IDF and Fast Text encodings. Same as in SVM section, ROC and AUC values of 10−fold cross validation for Random Forest are high as well. This means that our model works good. Standard deviation with respect to all runs is also low.

If we compare results for TF-IDF and Fast Text encodings, it is seen that Fast Text based model outperforms results with respect to TF-IDF encoding model. Mean AUC value for Fast Text article encoding model is higher by 5%. It is interesting that TF-IDF sparse matrix algorithm application in Random Forest ensembles model, results in almost the same computational time for Fast Text and TF-IDF encoding algorithms. This observation was made on the basis of processing 1000 articles from Comedy and Sports categories. As mentioned in SVM section we can also say that Random Forest algorithm is trained much faster than neural network based models.

It is possible to conclude that the model shows good results for this classification problem and can be used for active learning experiments.

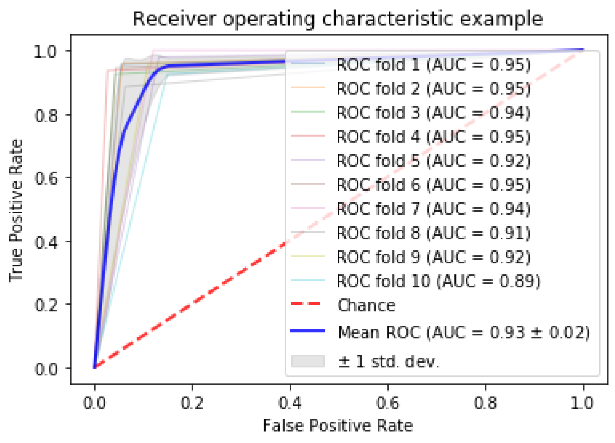

Figure 5.5: TF-IDF ROC and AUC for 10 neural networks ensembles trained and tested on Sports and Comedy data where each ensemble is trained on 70% of randomly chosen training data

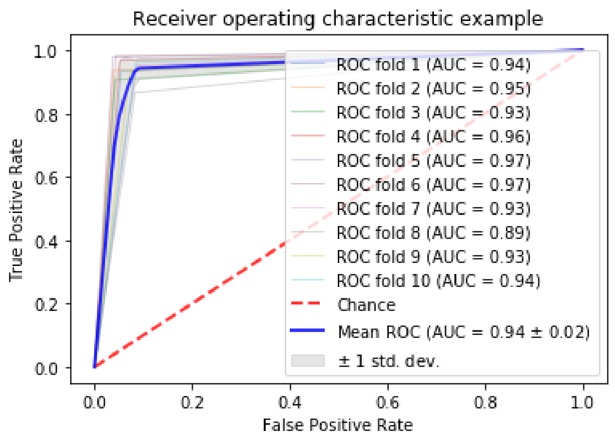

Figure 5.6: Fast Text ROC and AUC for 10 neural networks ensembles trained and tested on Sports and Comedy data where each ensemble is trained on 70% of randomly chosen training data

5.1.3 Feed Forward Neural Network Ensembles

In this work we have implemented three algorithms based on neural networks, such as neural networks ensembles, SGLD and DENFI algorithms. The difference between these methods is in parameters sampling. Neural networks ensembles takes a randomly selected subset from training data for each ensemble. Thus, we decided not to show SGLD and DENFI results in this section because they use the whole set of training data. Consequently, if neural networks ensembles show good results, we assume that SGLD and DENFI will also perform good passive learning outcome.

Results for neural networks ensembles are shown in figures 5.5 and 5.6. Illustrated results are both for TF-IDF and Fast Text encodings. As seen in previous section, ROC and AUC values of 10−fold cross validation for neural networks are high as well.

Comparing the results between TF-IDF and Fast Text encoding based models we can observe that Fast Text based model gives slightly better results. We have already discussed fastness of algorithm training with respect to different embedding models in Random Forest section. In case of neural networks, this difference is even more significant. The model that is based on Fast Text word embeddings takes approximately 20 times less elapsed time than TF-IDF encoding based model.

5.2 Conclusion

All of the tested models gave really good results in classification of Sports and Comedy categories. It means that we are able to continue working with these algorithms in active learning section. We would like to highlight that Fast Text encoding based algorithms showed a bit better results than TF-IDF. Fast Text based algorithms also showed significant improvement in evaluation speed of algorithm training.

Chapter 6

Active Learning Classification

In Passive Learning section we showed that all implemented algorithms are good for solving passive text classification task. Active Learning section results are the main part of this thesis. Therefore, we tested all five algorithms on the data mentioned in the Data section. The results are shown and described in further subsections. However, before starting with text classification results, we would like to show how our models represent uncertainty. As written in theoretical introduction to active learning, we use models' uncertainty for active learning loop.

6.1 Active Learning Models' Uncertainty

Due to the fact that text encoding features space dimensionality is extremely high, we introduce a 2-dimensional toy problem for uncertainty visualization. In figure 6.1 is shown a dataset that is used for toy problem classification. We generated this dataset with adding Gaussian noise in order to make the task similar to real world problems.

We generated 1000 data points where 500 are assigned to class 0 and another 500 points are assigned to class 1. We used 50% randomly chosen data samples as a training dataset. Next step, was creating a two dimensional grid that will be used for model predictions. We assign data sample to class 1 if prediction value is higher than 0.5. If prediction value is lower than 0.5, then the value is assigned to class 0. We consider that the model is uncertain about specific data sample if its prediction values is close

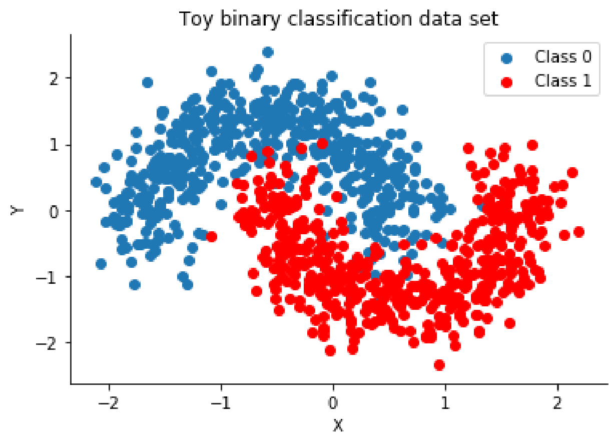

Figure 6.1: Toy problem dataset visualization

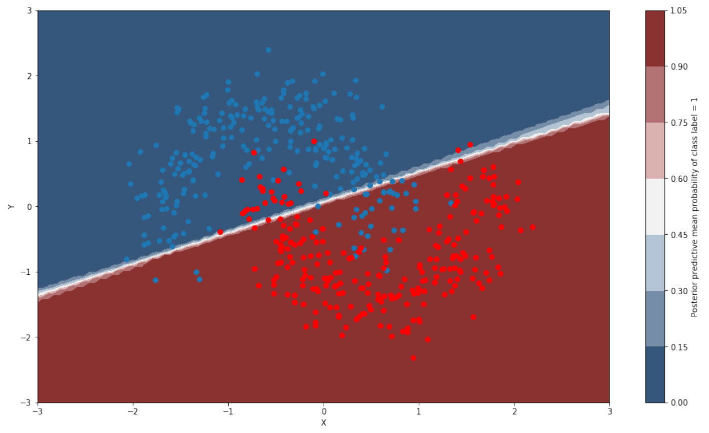

Figure 6.2: SVM ensembles posterior predictive mean probability of class 1

to 0.5. As a result, sampling values from maximal uncertainty region will bring maximum information about the dataset. Models set up for Toy Problem is the same as it is defined in the Machine Learning Component section 4.3 .

6.1.1 SVM Ensembles

Uncertainty for SVM ensembles model is visualized in figure 6.2. Uncertainty bounds are linear and quite narrow. Linearity of uncertainty bounds is explained with the fact that we are using SVMs with linear kernel. Narrowness can be explained with richness of the training dataset and linear limitations of SVM decision bound.

6.1.2 Random Forest Ensembles

In figure 6.3 is visualized Random Forest ensembles model uncertainty. In the case of Random Forest ensembles we can see that uncertainty bounds are not linear and lay near the region where two classes are splitted. We see that uncertainty region is becoming wider near the places where datapoint of two different classes lay close to each other. This is a behavior that we wanted to observe.

It is also seen that the curve is not smooth enough. This is caused by Random Forest decision boundary.

6.1.3 Neural Network Ensembles

In figure 6.4 is visualized neural network ensembles model uncertainty. In comparison to non-neural networks based methods that were shown above, uncertainty bounds are quite smooth. We can also observe that uncertainty bounds become wider when they go further from the data points. This behavior can be explained with the fact that in these places algorithm did not get any training data samples. We could also see this behavior in SVM ensembles, but in this case neural network models represent the uncertainties much better.

6.1.4 SGLD

In figure 6.5 is visualized SGLD model uncertainty. We see that SGLD algorithm has similar behavior to neural network ensembles. The difference is that all uncertainty bound curves has similar curvature.

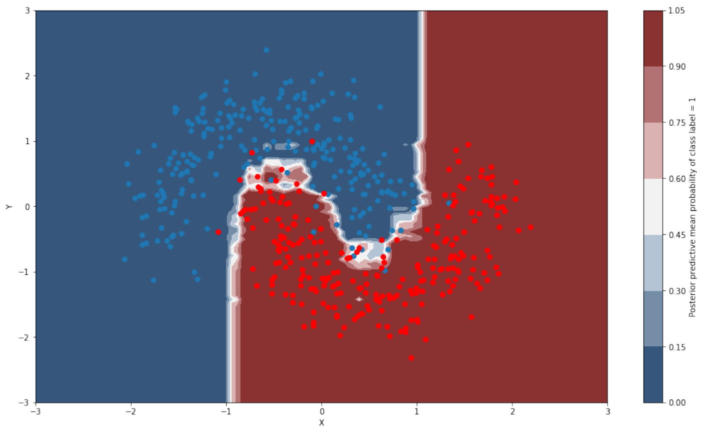

Figure 6.3: Random Forest ensembles posterior predictive mean probability of class 1

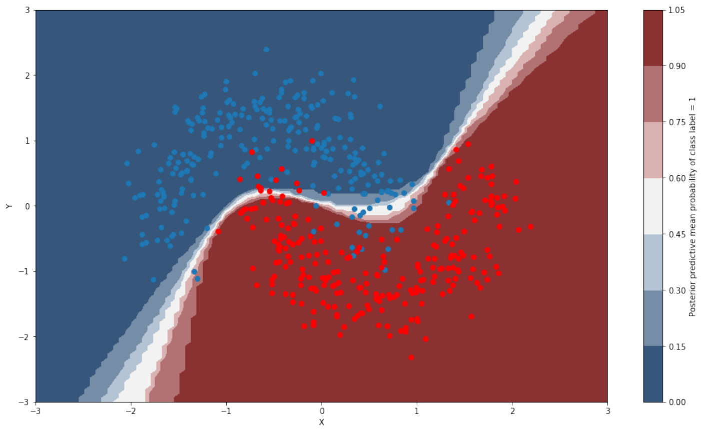

Figure 6.4: Neural network ensembles posterior predictive mean probability of class 1

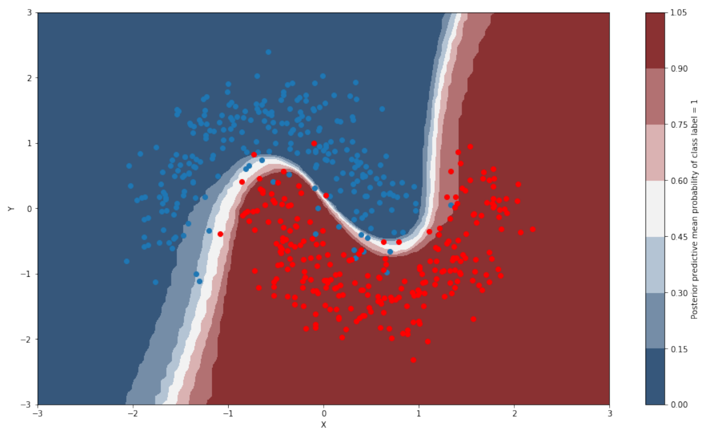

Figure 6.5: SGLD posterior predictive mean probability of class 1

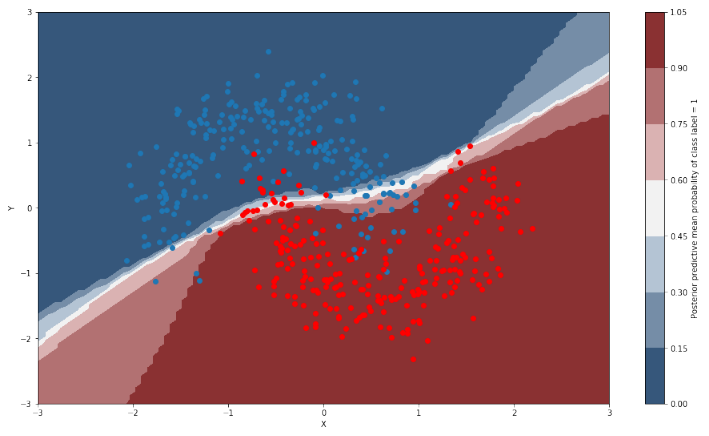

Figure 6.6: DENFI posterior predictive mean probability of class 1

This is happening due to the fact that SGLD finds loss function minimum and then samples parameters' values in a neighborhood of the minimum. As a result we expect decision bound for each parameters sample be quite similar to each other.

6.1.5 DENFI

In figure 6.6 is visualized DENFI model uncertainty. We see that uncertainty bounds are very similar to neural network ensembles algorithms but still a bit different. As told in pseudocode 4.1, algorithm founds different local minimums and then we add some Gaussian noise to parameters values. In further learning iterations the algorithm continues training using the weights from previous step.

The last 100 loss function values with respect to each DENFI ensemble are shown in figure 6.7. It is seen that each ensemble found its own local minimum. Additional Gaussian noise adds more variability to the algorithm that makes samples more diverse.

6.1.6 Conclusion

To conclude, we can say that the best variability representation was seen in neural network models. Obviously, we could test SVM algorithm with the usage of different kernels but our prior interest was

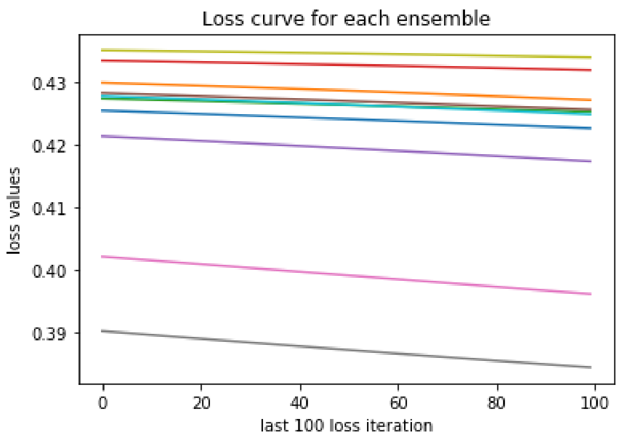

Figure 6.7: The last 100 loss function values with respect to each DENFI ensemble

concentrated on neural networks. We have showed that all models are able to represent variability and can be used for further tests.

6.2 Active Learning Simulation Set Up

6.2.1 Simulation Loop

The active learning simulation is done for comparing two strategies: i) random, and ii) ”smart” selection of text documents. We randomly choose initial training set that has 10 samples. Described 10 random samples are chosen from 1000 text documents (500 text documents per category). We reduce the size of all above mentioned datasets to 1000 documents. Each dataset is split on testing and training data.

Next we initialize two runs. First run is based on random selection of the text documents and second run is based on acquisition function selection of the documents. Both runs start with the same training dataset, and then they chose additional training documents based on their strategies. We consider continuous new data selection from 1000 documents dataset and imitating annotators labeling process. All in all, we imitate text samples selection 200 times ($U = 200$). We select 10 random samples in the beginning, train our model, and make a prediction on the complement to the training dataset (990 text documents from validation set). As a further step, we select 1 new sample from the set on which the prediction was done. New text sample for labeling is chosen with respect to acquisition function or random choice. Before we extend our training dataset with a new labeled sample, we calculate the AUC metrics on the complement to the dataset (990 text documents). This process is repeated 200 times. Thus, by the end of the simulation, our training set will have 210 text documents and testing set (complement to a training set) will have 790 data samples. In order to make our results statistically valid, we repeat the described simulation loop 10 times.

6.2.2 Epsilon Greedy Strategy

In this work we decided that we do not want to sample with respect to an acquisition function from the beginning but follow with epsilon greedy strategy [26]. We decided that both active learning and random strategy are going to start with random sampling. The algorithms have a coefficient $\epsilon \in [0, 1)$ that represents a probability of the data sampling with respect to the acquisition function. In this work

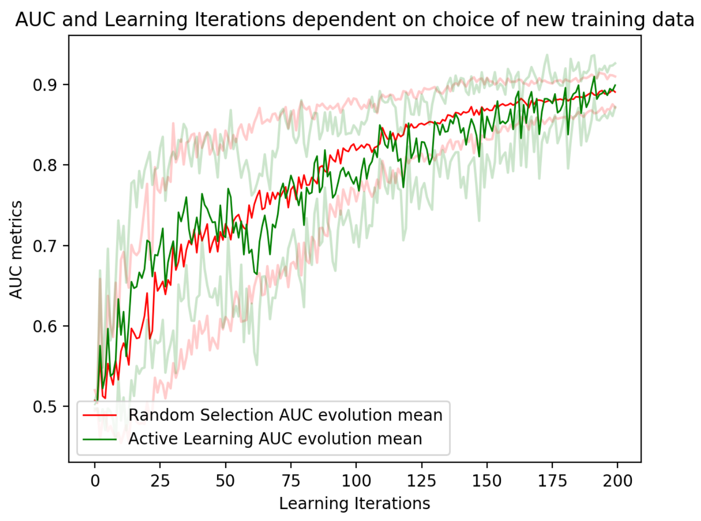

Figure 6.8: Comparison of random and active learning text data selection mean and one standard deviation for SVM ensembles with respect to 10 samples as initial training configuration, 200 data selection iterations and TF-IDF encoding

we define ϵ as

$$\epsilon = \begin{cases} \frac{\exp(u-40)}{\exp(u-40)+1}, & u \in \{1, ..., U\} \\ 0, & u = 0, \end{cases} \tag{6.1}$$

where $u = \{0, 1, ..., U\}$ is set of questions that results in number of text documents which will be labeled by an annotator. As mentioned in previous section, the number of iterations (questions U) is set to 200. From equation 6.1 is seen that when we reach 40−th document, the probability of random sampling is 50%. This strategy provides good results which will be shown in further sections.

6.3 Active Learning on Texts with TF-IDF Encoding Based Models

We decided to show the results for TF-IDF encoding based model because described text encoding provides high discriminability and relatively simple at the same time. However, as mentioned in passive learning section, due to the high amount of features models that are using this type of encoding are harder to train. We were not able to train neural network ensembles because of the high dimensional feature space. Thus, we decided to show the results only with respect to SVM, Random Forest and Sports, Comedy categories. Therefore, there is no need to split further section for different datasets because all the experiments based on TF-IDF encoding are assumed to be done only for Sports and Comedy categories.

6.3.1 SVM Ensembles

In figure 6.8 is shown active learning simulation results for SVM ensembles and TF-IDF encoding. Active learning strategy is based on entropy acquisition function.

As seen from figure 6.8, the results for entropy are not better than for random selection. This can be explained due to low variability of SVM ensembles. For this case we can conclude that there are

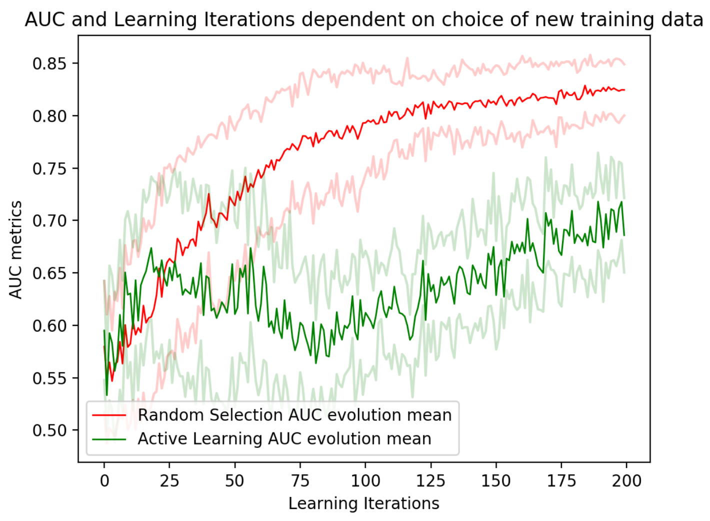

Figure 6.9: Comparison of random and active learning text data selection mean and one standard deviation for Random Forest ensembles with respect to 10 samples as initial training configuration, 200 data selection iterations and TF-IDF encoding

no significant difference between entropy and random data selection strategy. The first and the second strategy has almost same uncertainty bounds and converge to the same AUC results.

6.3.2 Random Forest Ensembles

In figure 6.9 is shown active learning simulation results for Random Forest ensembles and TF-IDF encoding. Active learning strategy is based on the entropy acquisition function.

In comparison to SVM ensembles, active learning strategy for Random Forest ensembles show really poor results. Random data selection overcomes entropy selection, and it can be said that active learning strategy is not working at all for this algorithm. The reason of such behavior can be explained due to the little amount of training data in the beginning. This fact may make algorithm to start selecting the data which have high entropy but are close to each other. As a result, it will not lead to high performance results.

6.3.3 Conclusion

Even though, we were able to see extremely good results for TF-IDF encoding in Passive Learning section, we were not able to train neural networks models and test active learning there, due to high dimensionality of the feature space. Moreover, SVM and Random Forest ensembles' results were unsatisfying, and in comparison to random sampling, active learning strategy did not show good results.

6.4 Active Learning on Texts with Fast Text Encoding Based Models

In comparison to section 6.3, in this section we show results to all models because Fast Text encoding maps texts to 300 dimensional feature space. This is much lower than in the case of TF-IDF encoding. Thus, we were able to provide computations with respect to all models.

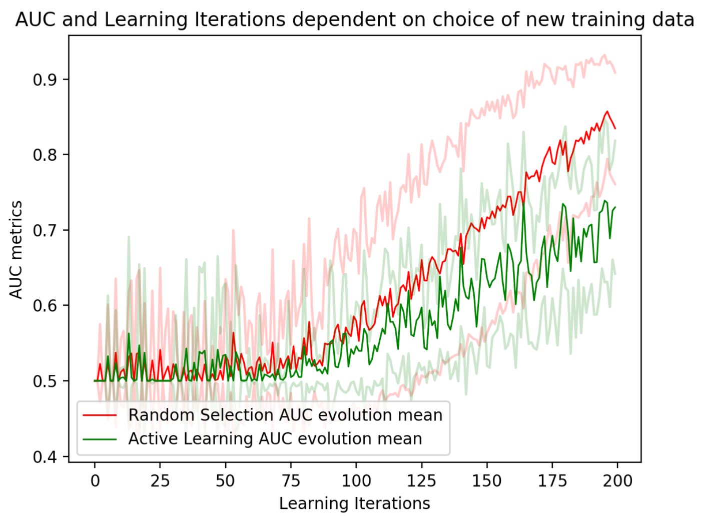

Figure 6.10: Comparison of random and active learning text data selection mean and one standard deviation for SVM ensembles with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Sports vs. Comedy categories

6.4.1 SVM Ensemble

6.4.1.1 Sports and Comedy Categories

In figure 6.10 is shown active learning simulation results for SVM ensembles and Fast Text encoding. Active learning strategy is based on the entropy acquisition function.

The behavior which we observe in figure 6.10 is similar to figure 6.8. We can see that active learning strategy is not working for the case of Fast Text encoding as well.

6.4.1.2 Conclusion

We can conclude that even despite good performance in Passive Learning section, the active learning algorithm is not working as expected. Thus, we will not continue testing this algorithm on further datasets.

6.4.2 Random Forest Ensembles

6.4.2.1 Sports and Comedy Categories

In figure 6.11 is shown active learning simulation results for Random Forest ensembles and Fast Text encoding. Active learning strategy is based on the entropy acquisition function.

As illustrated in figure 6.10 we can see the strategies results are similar to each other but in active learning case, entropy based sampling is overcoming random sampling strategy. We can also see that in active learning case uncertainty bounds are much more narrow.

6.4.2.2 Conclusion

Even though we could observe that active learning strategy shows better results than random sampling strategy, we do not continue with further experiments because the results are not significant enough.

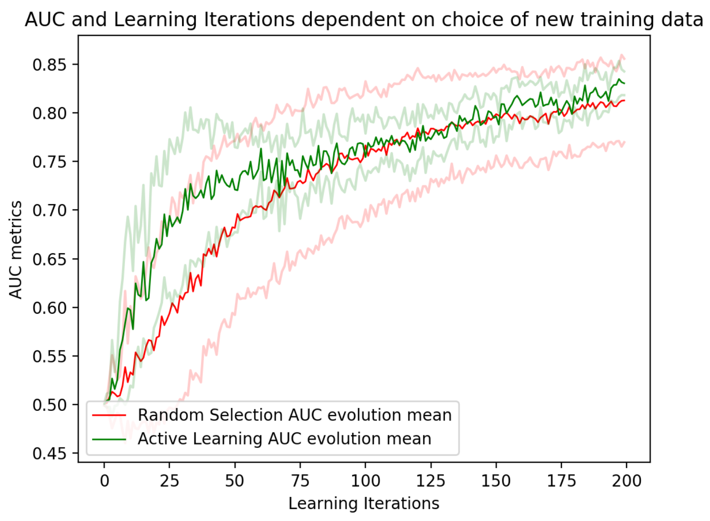

Figure 6.11: Comparison of random and active learning text data selection mean and one standard deviation for Random Forest ensembles with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Sports vs. Comedy categories

6.4.3 Neural Network Ensembles

6.4.3.1 Sports and Comedy Categories

In figure 6.12 is shown active learning simulation results for neural networks ensembles and Fast Text encoding. Active learning strategy is based on the entropy acquisition function.

If we compare output metrics from figure 6.12 and figures 6.10, 6.10, we can see significant difference in the algorithms performance. In comparison to previous methods, algorithm based on neural networks shows same results by $50-$th iteration. It means that the probability of using non-random acquisition function equals to 50%. We are able to observe significant improvement of entropy based sampling over the random sampling strategy. We can also see, that random sampling strategy is converging very slow to the results of the active learning strategy. Moreover, active learning uncertainty bounds are much more narrow than the uncertainty bounds of the random sampling algorithm.

6.4.3.2 Conclusion

One significant disadvantage of this method is that it is very slow and hard to train. Each time, we have to train 10 neural networks with cold start. That means training neural networks with random weights initialization and no prior information from the previous training. Despite the fact, that the results are really good, we decided not to continue with the algorithm testing because it is very slow and computationally costly. In next sections we introduce results based on neural networks but with different way of uncertainty representation.

6.4.4 SGLD

Stochastic Gradient Langevin Dynamics algorithm [27] is one of two algorithms that are assumed as the best algorithms (in terms of this project) with good ratio of training time consumption and performance. As mentioned previously, Gaussian noise is added to a Gradient Descent while training. This

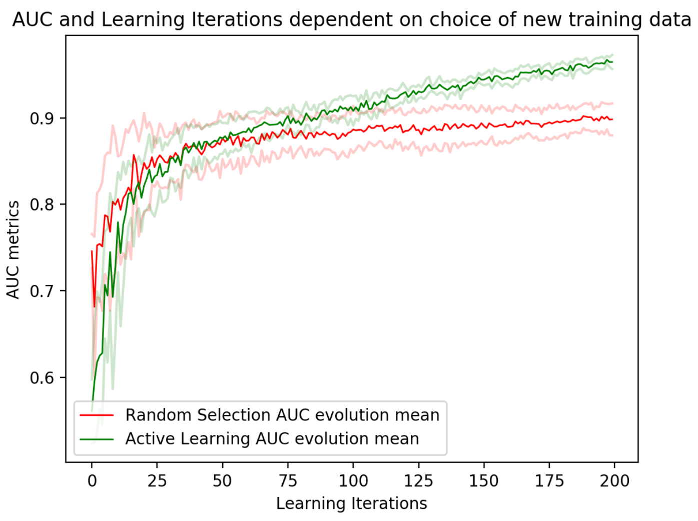

Figure 6.12: Comparison of random and active learning text data selection mean and one standard deviation for Feed Forward Neural Network ensembles with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Sports vs. Comedy categories

approach helps us to sample different parameters vectors around the loss minimum by simply continuing training. In comparison to neural networks ensembles, we do not have to train neural network ensembles separately, but train only one neural network with some additional training epochs.

6.4.4.1 Sports and Comedy Categories

In figure 6.13 is shown active learning simulation results for SGLD, Fast Text encoding and Sport vs. Comedy categories. Active learning strategy is based on the entropy acquisition function.

We can see that in figure 6.13 both mean and uncertainty bound curves are not smooth enough in comparison to neural network ensembles. This behavior can be explained with low number of samples from SGLD. However, we are able to observe same analogy as with neural network ensembles. The active learning strategy improves over random sampling and has more narrow uncertainty bounds. Moreover, SGLD was trained almost three times faster than the neural network ensembles algorithm.

6.4.4.2 Crime and Good News Categories

In figure 6.14 is shown active learning simulation results for SGLD, Fast Text encoding and Crime vs. Good News categories. Active learning strategy is based on the entropy acquisition function.

We expect Crime and Good News categories have small intersection. As seen from 6.14 it is really true, because the first iteration of the simulation starts at around $85\% - 90\%$ AUC. It means that only 10 training data samples algorithm could reach high performance results. The evolution of two strategies is quite the same as in figure 6.13. We would like to pay attention to the place where the upper uncertainty bound reaches the value that is greater than one. It is not possible for AUC to be greater than one because AUC $\in [0, 1]$. However, as mentioned previously, we construct uncertainty bounds as a standard deviation from mean value. In this case it is possible that upper uncertainty bound will be greater than one. Thus, while interpreting the results, reader has to keep in mind that AUC metric must be truncated to one.

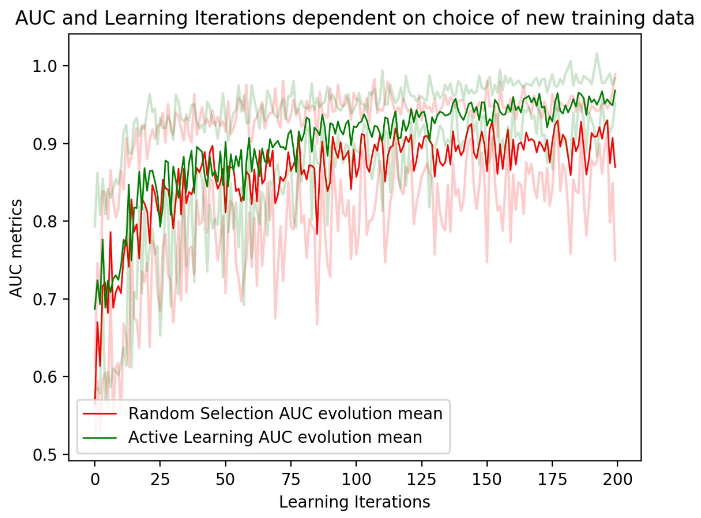

Figure 6.13: Comparison of random and active learning text data selection mean and one standard deviation for SGLD algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Sports vs. Comedy categories

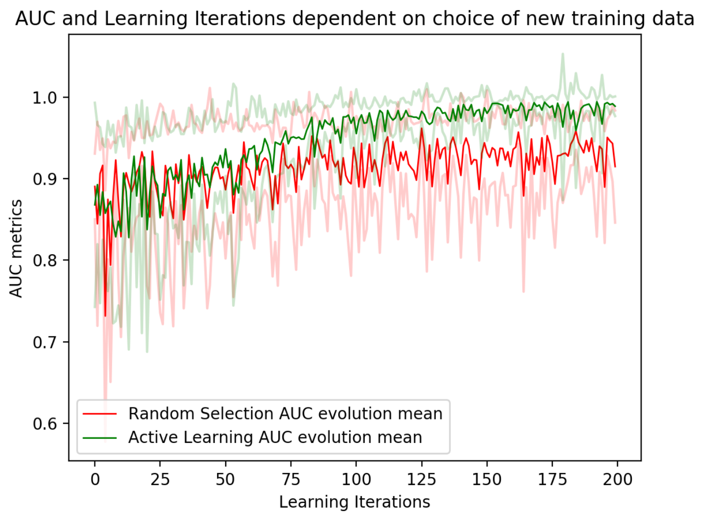

Figure 6.14: Comparison of random and active learning text data selection mean and one standard deviation for SGLD algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Crime vs. Good News categories

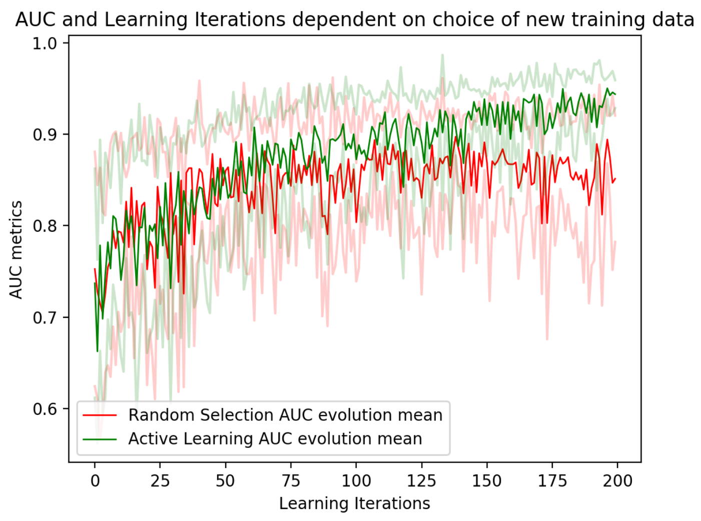

Figure 6.15: Comparison of random and active learning text data selection mean and one standard deviation for SGLD algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Politics vs. Business categories

6.4.4.3 Politics and Business Categories

In figure 6.15 is shown active learning simulation results for SGLD, Fast Text encoding and Politics vs. Business categories. Active learning strategy is based on the entropy acquisition function.

These two categories have bigger intersection, what means that it is a harder classification task. The results show again that active learning strategy easily exceeds the random sampling strategy. Even though, the topics are harder to distinguish, active learning strategy finds the patterns in the data and show much better results with more narrow uncertainty bounds.

6.4.4.4 Tech and Science Categories

In figure 6.16 is shown active learning simulation results for SGLD, Fast Text encoding and Tech vs. Science categories. Active learning strategy is based on the entropy acquisition function.

We observe again that active learning strategy is better even though the categories are quite similar. As seen from the plot uncertainty upper bound reaches the values which is greater than one. This is exactly the case which was already covered in 6.4.4.2.

6.4.4.5 College and Education Categories

In figure 6.17 is shown active learning simulation results for SGLD, Fast Text encoding and College vs. Education categories. Active learning strategy is based on the entropy acquisition function.

The collected results based on College and Education category do not differ to the results from previous parts of SGLD algorithm. We can still see that active learning strategy outperforms random sampling, even despite the fact that these categories are very similar.

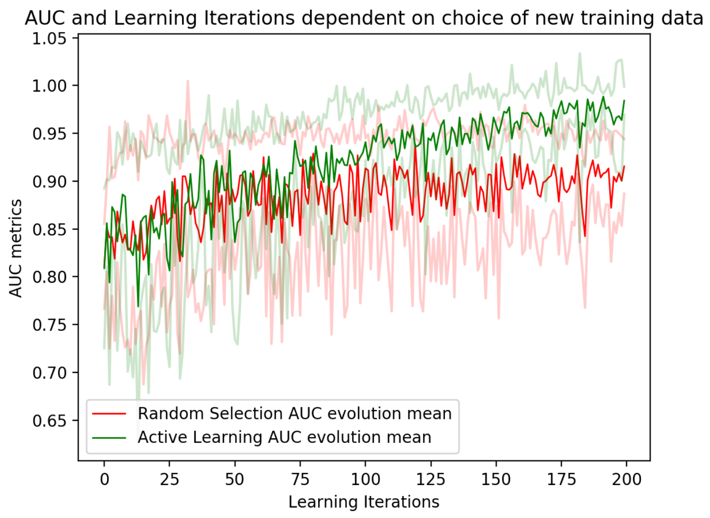

Figure 6.16: Comparison of random and active learning text data selection mean and one standard deviation for SGLD algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Tech vs. Science categories

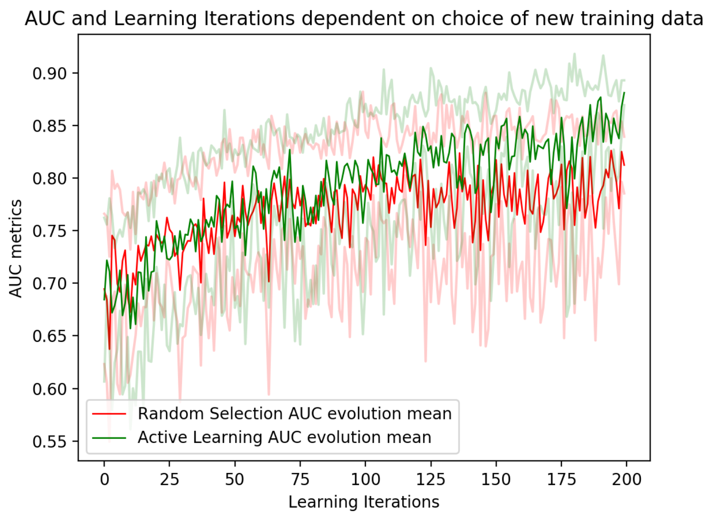

Figure 6.17: Comparison of random and active learning text data selection mean and one standard deviation for SGLD algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and College vs. Education categories

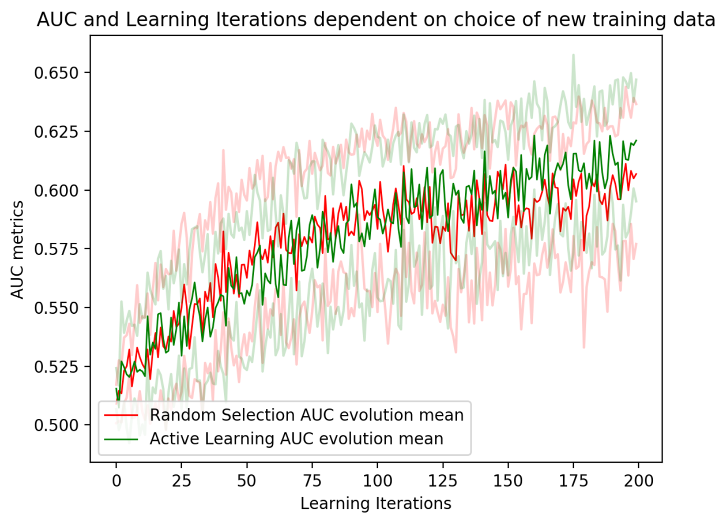

Figure 6.18: Comparison of random and active learning text data selection mean and one standard deviation for SGLD algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Positive vs. Negative tweets categories

6.4.4.6 Positive and Negative Tweets Categories

In figure 6.18 is shown active learning simulation results for SGLD, Fast Text encoding and Positive vs. Negative tweets categories. Active learning strategy is based on the entropy acquisition function.

In previous sections we tested SGLD algorithm on the data from HuffPost dataset. In figure 6.18 are illustrated results with respect to the tweets dataset. Due to the fact that tweets are really short, it is quite hard to find the patterns that can be used for high performance categories separation. Thus, we can see that our classification results (AUC) is not that high in comparison to previous results. Moreover, we observed that for this dataset there is no difference between active learning and random sampling strategy.

6.4.4.7 Conclusion

To sum up, we can say that SGLD algorithm showed really impressive results and proved that the active learning strategy outperforms random sampling. However, there are several disadvantages. First disadvantage is that the resulted curves were not smooth. We think that this problem can be eliminated by sampling more parameters vectors while training the model. Another problem is that the algorithm do not show better active learning results when it is tested on the short and quite general text data. We believe that this problem can be solved with different encoding approach.

6.4.5 DENFI

DENFI algorithm is the second algorithm which is not partitioning training dataset. In addition to this, we use a modification of DENFI algorithm. Huge advantage of this algorithm is that it uses prior information from previous training round. In the first training round, DENFI algorithms follows almost same strategy as neural network ensembles method. It trains 10 ensembles with respect to all training data. The variability in ensembles is reached with different weights initialization. After the training is finished, we add some Gaussian noise in order to increase the variability. When we add some training

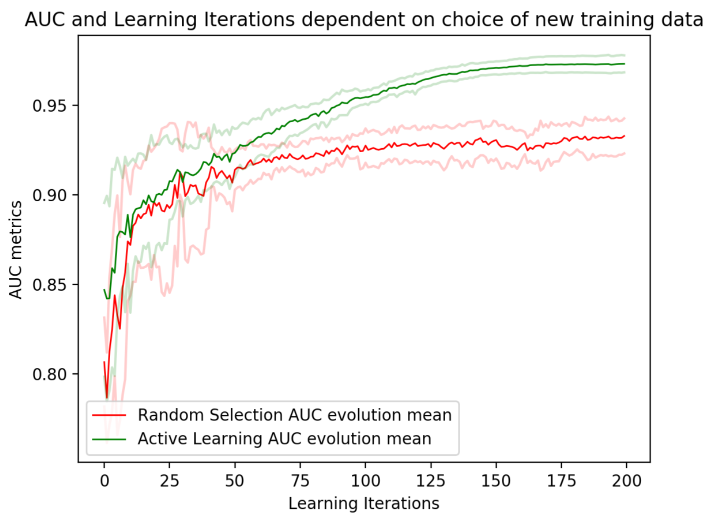

Figure 6.19: Comparison of random and active learning text data selection mean and one standard deviation for DENFI algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Sports vs. Comedy categories

samples, we continue training using the weights from the previous iteration with addition of Gaussian noise in the end. Moreover, we are using lower number of epochs. The hot start approach gives perfect variability that is shown in quality of results in further section.

6.4.5.1 Sports and Comedy Categories

In figure 6.19 is shown active learning simulation results for DENFI, Fast Text encoding and Sport vs. Comedy categories. Active learning strategy is based on the entropy acquisition function.

The active learning result, displayed in figure 6.19, outperforms all the algorithms that were tested before. We observe that the uncertainty bounds are extremely narrow. In addition to this, we see that active learning strategy has much higher AUC metrics and the curves are quite smooth. Moreover, due to the hot start training, the time needed to fit the algorithm is much lower in comparison to neural network ensembles.

6.4.5.2 Crime and Good News Categories

In figure 6.20 is shown active learning simulation results for DENFI, Fast Text encoding and Crime vs. Good News categories. Active learning strategy is based on the entropy acquisition function.

We have already talked in SGLD section that Crime and Good news categories can be well separated from each other. In this case we see same behavior for active learning strategy as in previous section. The uncertainty bounds are narrow and the AUC scores are very high in comparison to the random sampling strategy.

6.4.5.3 Politics and Business Categories

In figure 6.21 is shown active learning simulation results for DENFI, Fast Text encoding and Politics vs. Business categories. Active learning strategy is based on the entropy acquisition function.

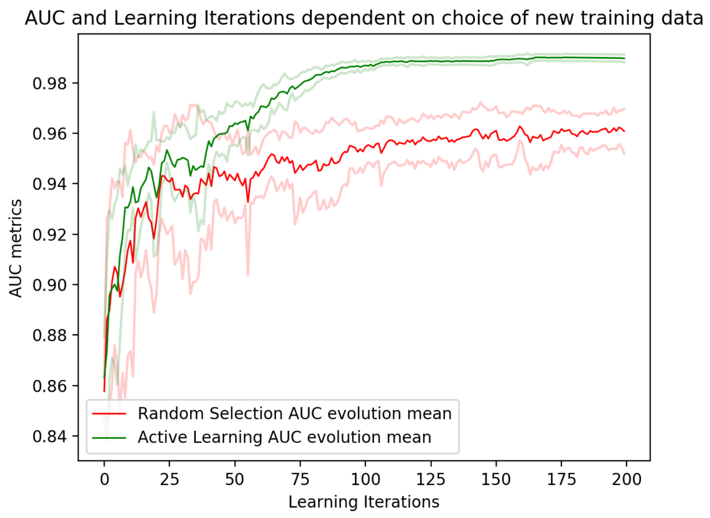

Figure 6.20: Comparison of random and active learning text data selection mean and one standard deviation for DENFI algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Crime vs. Good News categories

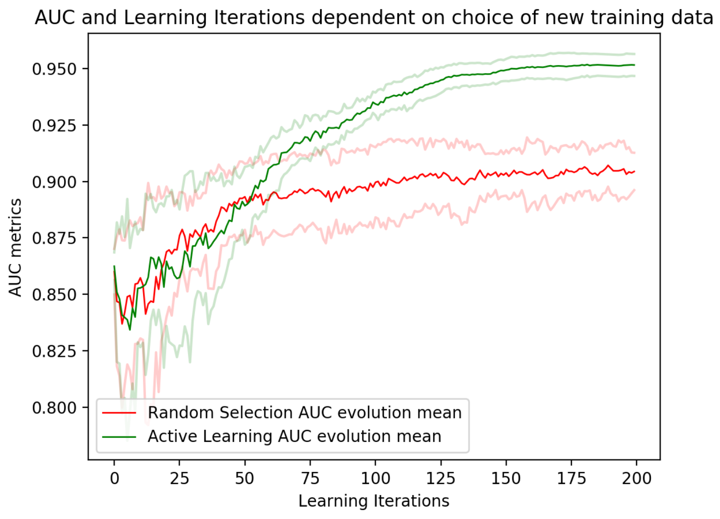

Figure 6.21: Comparison of random and active learning text data selection mean and one standard deviation for DENFI algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Politics vs. Business categories

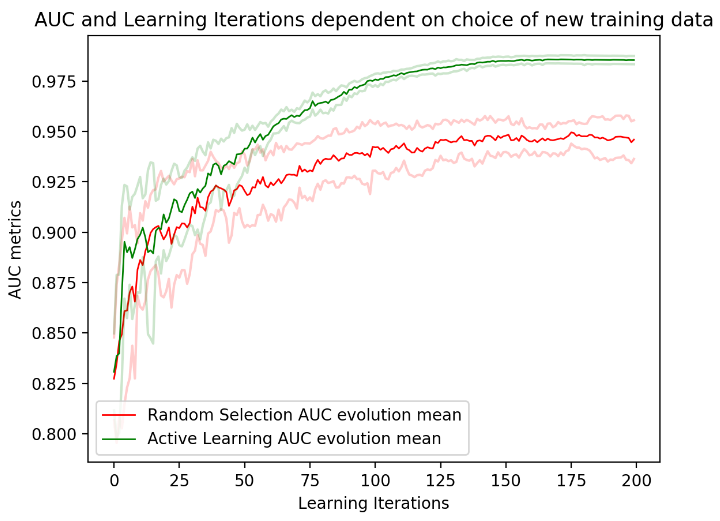

Figure 6.22: Comparison of random and active learning text data selection mean and one standard deviation for DENFI algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Tech vs. Science categories

The behavior, observed in figure 6.21, is same as in previous DENFI results. Moreover, we see good active learning performance, even though it is not easy to separate these categories. If we compare results in figure 6.21 to SGLD results in figure 6.15, we can see DENFI is better not only because its less corrupted with noise but also in higher AUC scores.

6.4.5.4 Tech and Science Categories

In figure 6.22 is shown active learning simulation results for DENFI, Fast Text encoding and Tech vs. Science categories. Active learning strategy is based on the entropy acquisition function.

The active learning strategy results for DENFI algorithm are much better in comparison to SGLD method that are shown in figure 6.16. We see that for active learning strategy case DENFI easily found the patterns which helped it to learn faster and show better scores with more narrow uncertainty bounds.

6.4.5.5 College and Education Categories

In figure 6.23 is shown active learning simulation results for DENFI, Fast Text encoding and College vs. Education categories. Active learning strategy is based on the entropy acquisition function.

Last but not least of Huffpost dataset, results for College and Education also proof that DENFI active learning strategy is not too much dependent on how big intersection is between the categories. The uncertainty bounds for active learning are still narrow and the AUC score is much higher than for random sampling case.

6.4.5.6 Positive and Negative Tweets Categories

In figure 6.24 is shown active learning simulation results for DENFI, Fast Text encoding and Positive vs. Negative tweets categories. Active learning strategy is based on the entropy acquisition function.

In comparison to SGLD, DENFI algorithm show better active learning performance for the tweets dataset. DENFI active learning strategy is also working on tweets whereas SGLD active learning strategy

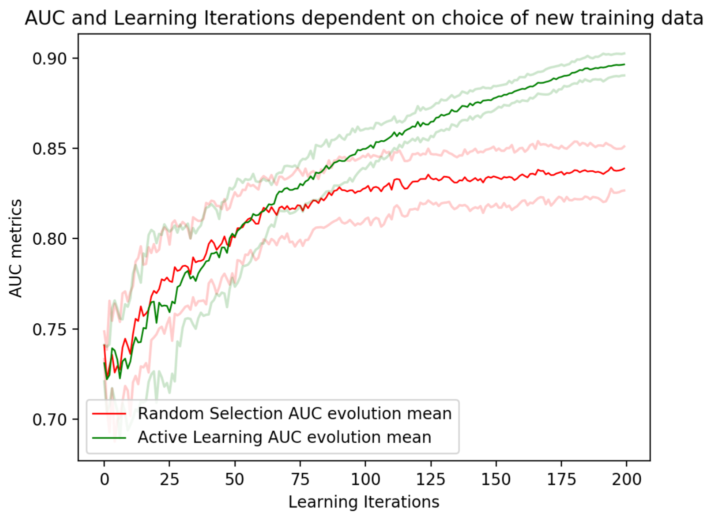

Figure 6.23: Comparison of random and active learning text data selection mean and one standard deviation for DENFI algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and College vs. Education categories

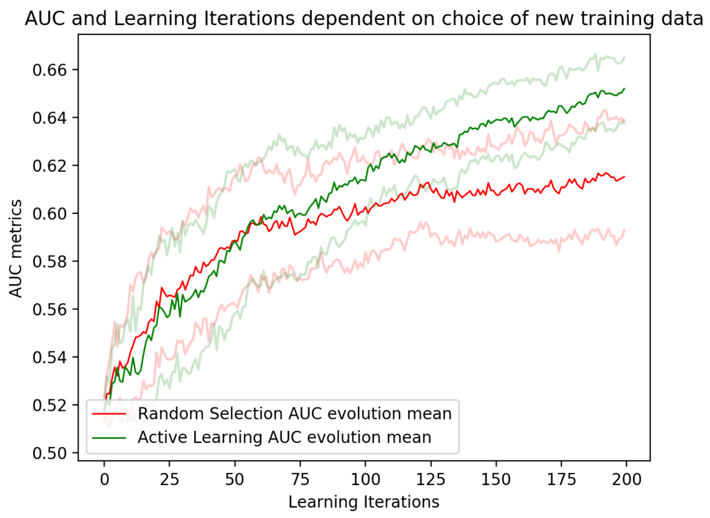

Figure 6.24: Comparison of random and active learning text data selection mean and one standard deviation for DENFI algorithm with respect to 10 samples as initial training configuration, 200 data selection iterations, Fast Text encoding and Positive vs. Negative tweets categories

was not able to overcome the random sampling. We can also see that DENFI AUC scores for active learning are higher with more narrow uncertainty bounds than for SGLD.

6.4.5.7 Conclusion

To conclude we can say that DENFI showed brilliant results for all problems which it was tested on. In all cases the active learning strategy outperformed the random sampling. The algorithm was tested both on the data that are easy and hard to separate. In addition to this, the AUC scores for all active learning results with DENFI were higher than for all other algorithms. That makes DENFI algorithm the best one, with the highest performance and the lowest uncertainty bounds, which was tested in term of this project.

Conclusion

This project shows how active learning strategy of querying unlabeled text documents for further labeling and training can beat random selection strategy. We have provided not only a high-level theoretical description of the problem but also testing results that cover different scenarios and text document categories. Github link for Python implementation is also available in this project.

Based on the achieved results that were gathered from testing on 12 different categories, we were able to see that a modification of the DENFI algorithm shows excellent performance and overcomes other algorithms in all aspects. DENFI outperforms all other models, both in higher AUC scores and more narrow uncertainty bounds. Another considerable advantage of DENFI is that it is the fastest neural network model that was implemented and tested in this thesis.

We see plenty of further opportunities for how it is possible to improve the algorithm, starting from better text representation and ending using other DENFI modifications. To conclude, we would like to say that the ensembles showed their power in solving an active learning problem, and in our opinion, it is a good field for continuing the research.

Bibliography

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

[2] Bang An, Wenjun Wu, and Huimin Han. Deep active learning for text classification. In *Proceedings of the 2nd International Conference on Vision, Image and Signal Processing*, pages 1–6, 2018.

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

[4] Christopher M Bishop. Machine learning and pattern recognition. *Information Science and Statistics. Springer, Heidelberg*, 2006.

[5] Sophie Burkhardt, Julia Siekiera, and Stefan Kramer. Semisupervised bayesian active learning for text classification. In *Bayesian Deep Learning Workshop at NeurIPS*, 2018.

[6] Shubhomoy Das, Md Rakibul Islam, Nitthilan Kannappan Jayakodi, and Janardhan Rao Doppa. Active anomaly detection via ensembles. *arXiv preprint arXiv:1809.06477*, 2018.

[7] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[8] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1183–1192. JMLR. org, 2017.

[9] Huang L. Go A., Bhayani R. Twitter sentiment classification using distant supervision, 2009.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413, 2017.

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

[13] David Lowell, Zachary C Lipton, and Byron C Wallace. Practical obstacles to deploying active learning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 21–30, 2019.

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

[15] 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.

[16] Rishabh Misra. News category dataset, 06 2018.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

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

[20] 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.

[21] Claude E Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[22] Yanyao Shen, Hyokun Yun, Zachary C Lipton, Yakov Kronrod, and Animashree Anandkumar. Deep active learning for named entity recognition. *arXiv preprint arXiv:1707.05928*, 2017.

[23] Jasper Snoek, Yaniv Ovadia, Emily Fertig, Balaji Lakshminarayanan, Sebastian Nowozin, D Sculley, Joshua Dillon, Jie Ren, and Zachary Nado. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, pages 13969–13980, 2019.

[24] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

[25] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[26] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[27] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.