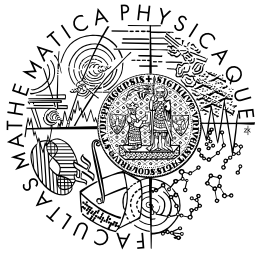


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Mitchell Borchers

Active learning in E-Commerce Merchant Classification using Website Information

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: Mgr. Marta Vomlelová, Ph.D.
Study programme: Artificial Intelligence
Study branch: IUIPA

Prague 2023

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor, Mgr. Marta Vomlelová, Ph.D., for her guidance throughout the duration of my thesis, as well as Adam Dobiáš and his team for the opportunity to work on this project. Additionally, I extend a big thank you to my parents, Daniel and Mary, and sisters for their unending support and encouragement.

Title: Active learning in E-Commerce Merchant Classification using Website Information

Author: Mitchell Borchers

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Marta Vomlelová, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Data and the collection and analysis of data plays an important role in everyday life even though it often goes unseen. In our case, our partner is using data to classify websites into different categories. We used active learning and other machine learning methods to help classify websites into these categories and to explore the data collection and classification process. We scraped text data from websites, translated the data to English, and then worked with machine learning tools to understand the data and classify it. We found that the xPAL active learning strategy and linear support vector classifiers seemed to perform best with our data.

Keywords: active learning xPAL machine learning multi-class classification website classification data mining web crawler web scraper

Contents

Introduction **3**

Definitions **5**

1 Active Learning **7**
1.1 Introduction . . . 7
1.2 Query Function Construction . . . 8
1.2.1 Pool-Based . . . 8
1.2.2 Stream-Based . . . 8
1.2.3 Membership Queries . . . 8
1.3 Sampling Strategies . . . 9
1.3.1 Random Sampling . . . 9
1.3.2 Diversity Sampling . . . 9
1.3.3 Uncertainty Sampling . . . 9
1.3.4 EER . . . 9
1.3.5 PAL . . . 10
1.3.6 xPAL . . . 10
1.3.7 ALCE . . . 10
1.3.8 QBC . . . 11
1.4 Classifiers . . . 11
1.5 Summary . . . 11

2 Defining xPAL **12**
2.1 Kernel . . . 12
2.2 Risk . . . 13
2.3 Conjugate Prior . . . 13
2.4 Risk Difference Using the Conjugate Prior . . . 14
2.5 Expected Probabilistic Gain . . . 15

3 Related Works **16**
3.1 E-commerce Merchant Classification . . . 16
3.2 Influence . . . 16

4 Data Review **17**
4.1 Overview . . . 17
4.2 Collection . . . 18
4.3 Processing . . . 18
4.4 Statistical Tests . . . 21
4.4.1 Chi-squared . . . 21
4.4.2 Variable Importance . . . 22
4.4.3 Fisher's Exact Test . . . 23

5 Testing with Original Data **24**
5.1 PWC, RBF Kernel, and Active Learning . . . 24
5.2 PWC, Cosine Kernel, and Active Learning . . . 24
5.3 Classifier Evaluation . . . 26

6 Testing with All Data **33**
6.1 Classifier Evaluation Revisited 33
6.2 Active Learning using All Data 33
6.3 LinearSVC with Text Length Filtering 36
6.4 Testing with Fewer Categories 37

Conclusion **40**

Bibliography **42**

List of Figures **44**

List of Tables **45**

A Attachments **46**

Introduction

One of the main challenges of creating a successful machine learning model is obtaining labeled data. With easy access to a variety of modern tools, devices, and sensors, we are able to rapidly collect unlabeled data. But, in supervised learning, prediction models are trained using labeled data. The problem is that acquiring labels for the collected data can be expensive, time-consuming, or even impossibly difficult in some cases.

However, methods have been developed to help reduce the number of labeled data required to train the classifier. Active learning is a semi-supervised machine learning framework where the model is trained with a smaller set of labeled data but which also aims to exploit trends within the unlabeled data. It's a framework in which the learner has the freedom to select which data points are added to its training set (Roy and McCallum [2001]).

Active learning is different from other frameworks because it uses the unlabeled data and some evaluation criteria to determine which candidate could be the most beneficial to the model if it was given a label. The model requests the label from some oracle that provides the label then it takes this new labeled data and rebuilds the classifier. We describe it as semi supervised active learning because the model is initially trained on both the labeled and unlabeled data, and then active learning is used to select the most informative examples for labeling.

In our case we will provide a set of labeled data to the active learning framework (or sampling strategy). The sampling strategy will assume all the data is unlabeled and then choose a candidate from the unlabeled data pool. Then the label is revealed and the classifier is updated using the new data. The newly labeled data is then added to the labeled pool and the process repeats.

We have some data (website urls) for some company or business that are given to us from our partner. From this data our partner currently utilizes human labor to browse the website and then label the url with a category (23 labels) and a subcategory (234+ tags, that branch from the main category but still have some relation). This is a repetitive and expensive task that could be supplemented using active learning.

To reduce the amount of data required to train the classifier we consider a combination of tools and frameworks, namely: Scrapy, Postgres, a translation service, and an active learning sampling strategy paired with a classifier. We also explore the use of different classifiers to determine if there is some optimal classifier.

A website is required, then we use the Scrapy framework to navigate to the webpage, and collect then store the scraped data into the database. Next we access the data, translate the text, and add the translated data back into the database. During this process we also remove the html and numbers.

Once the the data is close to just pure text we use TF-IDF to transform it into a vectorized representation so we can use it with the classifiers. We experiment with different classifiers to determine if there is some optimal classifier for our data.

In the first section we introduce active learning and the different components of active learning. In the second section we look more into the details of xPAL

and how it works. In the third section we discuss the data and the steps we took to collect and process the data. In the fourth and fifth sections we conduct a variety of experiments to explore the performance of the sampling strategies and alternative classifiers.

Our goal is to understand the entire process including the web scraping, translation, storage, and performance of the selection strategies and classifiers. This analysis will allow our partner to learn from our tests and experiments. It will also allow them to make an informed decision on which models and selection strategies may be best suited for their needs moving forward.

Definitions

In this section we define some terms and ideas that will be helpful in understanding the upcoming sections.

Definition 1 (Beta Prior)**.** *A beta prior is a conjugate prior for the binomial distribution. It is a continuous probability distribution defined on the interval [0, 1] and is parameterized by two positive shape parameters, α and β. The beta distribution is defined as:*

$$Beta(\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

where Γ is the gamma function and x is a random variable. The gamma function is defined as:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

The gamma function is used as a normalizing constant to ensure that the probability density function integrates to 1 over the simplex, which is the space of all probability vectors that sum to 1.

Definition 2 (Conjugate Prior)**.** *A conjugate prior is a prior distribution that is in the same family of distributions as the likelihood function. In other words, the posterior distribution will have a similar functional form to the prior distribution.*

Definition 3 (Decision-Theoretic)**.** *Decision-theoretic active learning is a framework that uses the expected performance gain of a candidate to determine which candidate to label. The expected performance gain is the expected performance of the classifier after labeling the candidate minus the expected performance of the classifier before labeling the candidate. The expected performance of the classifier is the expected value of the performance measure given the posterior distribution of the classifier.*

Definition 4 (Dirichlet Distribution)**.** *The Dirichlet distribution is a multivariate generalization of the beta distribution. It is a continuous probability distribution defined on the K-simplex, $\Delta_K = \{x \in \mathbb{R}^K : x_i \geq 0, \sum_{i=1}^K x_i = 1\}$. The Dirichlet distribution is parameterized by a vector of positive shape parameters, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_K)$. The Dirichlet distribution is defined as:*

$$Dir(\alpha) = \frac{\Gamma(\sum_{i=1}^K \alpha_i)}{\prod_{i=1}^K \Gamma(\alpha_i)} \prod_{i=1}^K x_i^{\alpha_i - 1}$$

where Γ is the gamma function as defined in Definition 1 and where x is a random vector.

Definition 5 (Ground Truth)**.** *Ground truth is the true label of a data point.*

Definition 6 (Posterior Probabilities)**.** *Posterior probability is a type of conditional probability that results from updating the prior probability with information summarized by the likelihood via an application of Bayes' rule. The posterior probability is the probability of an event occurring given that another event has occurred.*

Definition 7 (Omniscient Oracles)**.** *Omniscient oracle is a hypothetical entity that has complete knowledge of the true labels of all data points in a given dataset. An omniscient oracle knows the ground truth labels of all data points.*

Definition 8 (TF-IDF)**.** *TF-IDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining. The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus (large structured set or collection of speech or text data).*

1. Active Learning

1.1 Introduction

Russel and Norvig succinctly define an agent and different types of learning in their book "Artificial Intelligence: A Modern Approach" (Russell and Norvig [2009]), their definition is paraphrased here. They define an agent as something that acts and a rational agent as one that acts so as to achieve the best outcome. If there is uncertainty, then the agent tries to achieve the best expected outcome. Any component of an agent can be improved by learning from data. The improvements and techniques used to make them depend on four major factors:

1. Which component is to be improved.
2. What prior knowledge the agent already has.
3. What representation is used for the data and the component.
4. What feedback is available to learn from.

Here we will mostly be focused on the final point, "What feedback is available to learn from". However, we will also discuss the importance of the second and third points because we will use Bayesian learning. There are three main types of feedback that determine the three main types of learning, which are: unsupervised, reinforcement, and supervised.

In unsupervised learning an agents goal is to discover patterns in the data even though no feedback or labels are provided. In reinforcement learning, the agent learns from a series of rewards or punishments that are dealt out based on its decisions. In supervised learning, an agent learns from input-output pairs, which can be discrete or continuous, to find a function that maps the pairs as best as possible.

The goal of supervised learning, given a training set of N example input-output pairs:

$$(x_1, y_1), (x_2,y_2),...(x_N,y_N),$$

where each y_j was generated by some unknown function $y = f(x)$, is to find a function h that approximates the true function f.

In reality, the types of learning overlap. In semi-supervised learning, some data points are labeled, and some are not. The model is trained on the labeled data, and then the knowledge gained from that labeled data is used to improve the model's predictions on the unlabeled data.

Supervised learning models almost always get more accurate with more labeled data. Active learning is the process of deciding which data to select for annotation (Munro [2021]). In other words, the central component of an active learning algorithm is the selection strategy, or deciding which of the unlabeled data could be the most useful to the model if it was labeled. Active learning uses a selection strategy that augments the existing classifier, it is not itself a classifier but rather an evaluation methodology working with a classifier.

Many different sampling strategies exist. First we will discuss query functions then we will briefly define three basic sampling strategies: uncertainty, diversity, and random sampling to get an idea of sampling. We will then discuss some other more advanced sampling strategies that are used in our experiments. When sampling the unlabeled data an ordered list is returned and the top candidate is the candidate that is expected to be most valuable for the model, but we are not strictly limited to taking just one candidate.

1.2 Query Function Construction

There are various techniques used to construct the querying functions. We will focus on pool-based active learning, but a number of interesting and relevant ideas appear within other active-learning frameworks that are worth mentioning.

1.2.1 Pool-Based

In pool-based active learning, a fixed set of unlabeled examples is provided at the start of the learning process, and the active learner iteratively selects a subset of these examples for annotation (Huang and Lin [2016]). The selection of the subset is based on a query strategy that aims to maximize the information gain from each annotation. Pool-based active learning is useful in situations where all the data is available in advance, such as in document classification or image classification.

1.2.2 Stream-Based

In stream-based active learning, data arrives in a continuous stream, and the active learner must make real-time decisions about which examples to label (Baram et al. [2004]). This is common in settings such as sensor networks or social media feeds. The selection of examples for annotation is based on a query strategy that takes into account the current state of the model, as well as the uncertainty and informativeness of each incoming example. The stream-based model can be viewed as an online version of the pool-based model.

1.2.3 Membership Queries

In membership query based active learning, the active learner can make queries to an oracle or construct a point in input space and requests its label from an oracle, such as a human expert, to obtain labels for specific examples (Baram et al. [2004]). The goal is to select the examples for which obtaining a label is most informative, in order to minimize the number of queries required to achieve a high accuracy. Membership-query-based active learning is useful when labeling each example is expensive or time-consuming, such as in medical diagnosis or legal document review.

1.3 Sampling Strategies

Sampling strategies, also referred to as selection strategies, are the core of the active learning process. The goal of sampling is to select the most useful data points from the unlabeled pool to label. The most useful data points are those that are expected to improve the classifier the most.

1.3.1 Random Sampling

Random sampling is self explanatory as it randomly selects an unlabeled data point from the pool and requests to have it labeled then it uses this newly selected data point to update the model. Random sampling is good to use as a baseline to compare other sampling strategies with.

1.3.2 Diversity Sampling

Diversity sampling is the set of strategies for identifying unlabeled items that are underrepresented or unknown to the machine learning model in its current state (Munro [2021]). The items may have features that are unique or obscure in the training data, or they might represent data that are currently underrepresented in the model.

Either way this can result in poor or uneven performance when the model is applied or the data is changing over time. The goal of diversity sampling is to target new, unusual, or underrepresented items for annotation to give the algorithm a more complete picture of the problem space.

1.3.3 Uncertainty Sampling

Uncertainty sampling is based on the idea that the most informative examples to query are the ones that the current model is most uncertain about. For example, in binary classification, an uncertain example might be one that is close to the decision boundary, or one that has a low predicted probability for the majority class (Munro [2021]). The idea is that by querying these uncertain examples, the model can better learn the boundary between the classes and improve its accuracy. Uncertainty sampling is simple given a classifier that estimates $P(C|w)$ (Lewis and Gale [1994]). On each iteration, the current version of classifier can be applied to each data point, and the data with estimated $P(C|w)$ values closest to 0.5 are selected, since 0.5 corresponds to the classifier being most uncertain of the class label.

These items are most likely to be wrongly classified, so they are the most likely to result in a label that differs from the predicted label, moving the decision boundary after they have been added to the training data and the model has been retrained.

1.3.4 EER

Monte Carlo estimation of error reduction (EER) estimates future error rate by log-loss, using the entropy of the posterior class distribution on a sample of the

unlabeled examples, or by 0-1 loss, using the posterior probabilities of the most probable class for the sampled unlabeled examples (Roy and McCallum [2001]).

Basically, the goal is to estimate the expected reduction in error for each unlabeled example by randomly sampling from the model's predictions and comparing the performance of the model with and without the example included in the training data.

1.3.5 PAL

Probabilistic Active Learning (PAL) follows a smoothness assumption and models for a candidate instance both the true posterior in its neighborhood and its label as random variables (Krempl et al. [2014]). By computing for each candidate its expected gain in classification performance over both variables, PAL selects the candidate for labeling that is optimal in expectation. PAL shows comparable or better classification performance than error reduction and uncertainty sampling, has the same asymptotic linear time complexity as uncertainty sampling, and its faster than error reduction based on the tests from the paper.

1.3.6 xPAL

Extended probabilistic gain for active learning (xPAL) is a decision-theoretic selection strategy that directly optimizes the gain and misclassification error, and uses a Bayesian approach by introducing a conjugate prior distribution to determine the class posterior to deal with uncertainties (Kottke et al. [2021]). Although the data distribution can be estimated, there is still uncertainty about the true class posterior probabilities.

These class posterior probabilities can be modeled as a random variable based on the current observations in the dataset. For this model, a Bayesian approach is used by incorporating a conjugate prior to the observations. This produces more robust usefulness estimates for the candidates.

1.3.7 ALCE

Active Learning with Cost Embedding (ALCE) is a non-probabilistic uncertainty sampling algorithm for cost-sensitive multiclass active learning (Huang and Lin [2016]). First a cost-sensitive multiclass classification algorithm called cost embedding (CE) was designed, which embeds the cost information in the distance measure in a special hidden space by non-metric multidimensional scaling. Then a mirroring trick was used to let CE embed the possibly asymmetric cost information in the symmetric distance measure.

It works by augmenting the example space with an additional dimension that represents the cost of labeling each example. This cost embedding can be learned from previous labeling efforts or estimated based on domain knowledge. The cost embedding can then be used to guide the active learning process by selecting examples that are not only informative but also cost-effective to label.

1.3.8 QBC

Query By Committee (QBC) uses an ensemble of classifiers that are trained on bootstrapped replicates of the labeled set (Seung et al. [1992]). The idea is to train a committee of classifiers on the available labeled data and then use the committee to select the most informative unlabeled data for labeling (Freund et al. [1997]). The committee consists of several classifiers, each trained on a slightly different subset of the available labeled data.

The QBC algorithm measures the disagreement of the committee's predictions on each unlabeled data point. The intuition is that if the committee members disagree then it is likely to be a difficult data point for the current classifier and thus informative for labeling.

The algorithm selects a fixed number of the most informative examples and requests their labels. The labeled examples are then added to the labeled pool, and the committee is retrained on the expanded labeled pool. This process is repeated until the algorithm achieves a desired level of accuracy or the available labeling budget is exhausted.

1.4 Classifiers

The classifier integrated into the active learning sampling strategy repository we used is the Parzen Window Classifier (PWC). It is a non-parametric method used for classification and density estimation in machine learning. It works by estimating the probability density function of a given class using a kernel density estimator, and then using Bayes' theorem to classify new instances based on their estimated probability densities.

We will also explore using other classifiers from Scikit-Learn and TensorFlow and compare their performance on the data without using active learning to see if there is any improvement beyond the PWC classifier.

1.5 Summary

It should now be more clear how the sampling strategy is the major component of active learning. The query function construction is also important but it is just a means of routing the data to be sampled. In the next chapter we will look into the specifics of xPAL.

2. Defining xPAL

We have introduced many different active learning sampling strategies in the previous section, and we will use them to test which strategy performs best with our data. However, we will mainly focus on using the xPAL sampling strategy and a pool based query function. The xPAL sampling strategy is a decision-theoretic approach to measure the usefulness of a labeling candidate in terms of its expected performance gain (Kottke et al. [2021]). We can estimate the data distribution but we are uncertain about the true class posterior probabilities. The class posterior probabilities are modeled as a random variable based on the current observations. Therefore a Bayesian approach is used by incorporating a conjugate prior to the observations. In general, the idea is to estimate the expected performance gain for the classifier, using the unlabeled data, and then select the best data point and request or reveal its label. Descriptions of the variables used throughout the paper are listed in Table 2.1.

	Descriptions
C	Number of classes
x	Input $x \in \mathbb{R}^D$ (D-dimensional vector space)
y	Class label $y \in \mathcal{Y}$
$\mathcal{Y}$	Set of all labels $\mathcal{Y} = \{1,...,C\}$
$f^{\mathcal{L}}$	Classifier that maps input x to label y using $\mathcal{L}$
L	Loss
R	Risk
$R_{\mathcal{E}}$	Empirical risk
$\mathcal{L}$	Set of labeled data $\{(x_1,y_1),...,(x_n,y_n)\}$
$\mathcal{U}$	Set of unlabeled data $\{x_1,...,x_n\}$
$\mathcal{E}$	Set of available labeled and unlabeled data $\{x : (x,y) \in \mathcal{L}\} \cup \mathcal{U}$

Table 2.1: Variable names and descriptions.

2.1 Kernel

A kernel based classifier is used in xPAL which determines the similarity of two data points. The kernel function $\boldsymbol{K}(x,x')$ is a function that maps two data points to a real number, which is then used to estimate the probability density of the data. The kernel frequency estimate $\boldsymbol{k}_x^{\mathcal{L}}$ of an instance x is calculated using the labeled instances $\mathcal{L}$. The y-th element of that C-dimensional vector describes the similarity-weighted number of labels of class y.

$$\boldsymbol{k}_{x,y}^{\mathcal{L}} = \sum_{(x',y') \in \mathcal{L}} \mathbb{1}_{y=y'} \boldsymbol{K}(x,x') \tag{2.1}$$

The Parzen Window Classifier uses the labeled data for training and predicts the most frequent class and was selected by Kottke et al. to use because of its

speed and ability to implement different kernels depending on the data (Kottke et al. [2021]). It was used for all the selection strategies in their experiments.

$$f^{\mathcal{L}}(x) = \arg\max_{y \in \mathcal{Y}} \left(\boldsymbol{k}^{\mathcal{L}}_{x,y} \right). \tag{2.2}$$

We will use the PWC classifier for our experiments because that is what is implemented with the active learning strategies, but we will also evaluate other classifiers and compare their performance less active learning.

2.2 Risk

For xPAL, Kottke et al. use the classification error as the performance measure and minimize the zero-one loss. The risk describes the expected value of the loss relative to the joint distribution given some classifier. The zero-one loss returns 0 if the prediction from the classifier is equal to the true class else it returns 1. The risk is a theoretical concept that cannot be computed directly since it requires knowledge of the entire population distribution. Instead, we typically try to approximate the risk using the empirical risk.

$$R(f^{\mathcal{L}}) = \mathop{\mathbb{E}}_{p(x,y)} [L(y, f^{\mathcal{L}}(x))] \tag{2.3}$$

$$= \mathop{\mathbb{E}}_{p(x)} \left[\mathop{\mathbb{E}}_{p(y|x)} [L(y, f^{\mathcal{L}}(x))] \right] \tag{2.4}$$

$$L(y, f^{\mathcal{L}}(x)) = \mathbb{1}_{f^{\mathcal{L}}(x) \neq y} \tag{2.5}$$

Because it is not known how the data is generated Kottke et al. use a Monte-Carlo integration with all available data $\mathcal{E}$ to represent the generator. The empirical risk $R_{\mathcal{E}}$ is the average of the loss over all data in the dataset. It refers to the average value of a given loss function over a finite set of observed data points.

$$R_{\mathcal{E}}(f^{\mathcal{L}}) = \frac{1}{|\mathcal{E}|} \sum_{x \in \mathcal{E}} \mathop{\mathbb{E}}_{p(y|x)} \left[L(y, f^{\mathcal{L}}(x)) \right] \tag{2.6}$$

$$= \frac{1}{|\mathcal{E}|} \sum_{x \in \mathcal{E}} \sum_{y \in \mathcal{Y}} p(y|x) L(y, f^{\mathcal{L}}(x)) \tag{2.7}$$

The empirical risk is a computable quantity that can be used as an estimate of the risk. However, it is only an approximation and is subject to sampling error.

2.3 Conjugate Prior

The conditional class probability $p(y|x)$ depends on the ground truth which is unknown. The conditional class probability is exactly the y-th element of the unknown ground truth vector $\boldsymbol{p}$. The ground truth is an unknown but fixed deterministic function $t : \mathbb{R}^D \to [0,1]^C$ that maps an instance x to a probability vector $\boldsymbol{p}$. The nearby labels from $\mathcal{L}$ can be used to estimate the ground truth $\boldsymbol{p}$ because the oracle provides the labels according to $\boldsymbol{p}$. If we assume a smooth

distribution then the estimate is relatively close to the ground truth if we have enough labeled instances.

$$p(y|x) = p(y|t(x)) = p(y|\boldsymbol{p}) = \text{Cat}(y|\boldsymbol{p}) = p_y \tag{2.8}$$

A Bayesian approach is used for estimation by calculating the posterior predictive distribution (calculating the expected value over all possible ground truth values). The probability of y given some x is approximately equal to the kernel frequency estimate of x.

$$p(y|x) \approx p(y|\boldsymbol{k}_x^{\mathcal{L}}) = \mathop{\mathbb{E}}_{p(\boldsymbol{p}|\boldsymbol{k}_x^{\mathcal{L}})}[p_y] = \int p(\boldsymbol{p}|\boldsymbol{k}_x^{\mathcal{L}}) p_y d\boldsymbol{p} \tag{2.9}$$

Bayes theorem is then used to determine the posterior probability of the ground truth at instance x in Equation 2.10. The likelihood $p(\boldsymbol{k}_x^{\mathcal{L}}|p)$ is a multinomial distribution because each label has been drawn from $\text{Cat}(y|\boldsymbol{p})$. A prior is introduced and selected as a Dirichlet distribution with $\alpha \in \mathbb{R}^C$ as this is the conjugate prior of the multinomial distribution. An indifferent prior is chosen and each element of alpha is set to the same value. The Dirichlet distribution is an analytical solution for the posterior when the conjugate prior of the multinomial likelihood are used.

$$p(\boldsymbol{p}|\boldsymbol{k}_x^{\mathcal{L}}) = \frac{p(\boldsymbol{k}_x^{\mathcal{L}}|\boldsymbol{p})p(\boldsymbol{p})}{p(\boldsymbol{k}_x^{\mathcal{L}})} \tag{2.10}$$

$$= \frac{\text{Mult}(\boldsymbol{k}_x^{\mathcal{L}}|\boldsymbol{p}) \cdot \text{Dir}(\boldsymbol{p}|\alpha)}{\int \text{Mult}(\boldsymbol{k}_x^{\mathcal{L}}|\boldsymbol{p}) \cdot \text{Dir}(\boldsymbol{p}|\alpha) d\boldsymbol{p}} \tag{2.11}$$

$$= \text{Dir}(\boldsymbol{p}|\boldsymbol{k}_x^{\mathcal{L}} + \alpha) \tag{2.12}$$

The conditional class probability is determined next from Equation 2.9. It is calculated with the expected value of the Dirichlet distribution.

$$p(y|\boldsymbol{k}_x^{\mathcal{L}}) = \mathop{\mathbb{E}}_{\text{Dir}(\boldsymbol{p}|\boldsymbol{k}_x^{\mathcal{L}}+\alpha)}[p_y] \tag{2.13}$$

$$= \int \text{Dir}(\boldsymbol{p}|\boldsymbol{k}_x^{\mathcal{L}} + \alpha) p_y d\boldsymbol{p} \tag{2.14}$$

$$= \frac{(\boldsymbol{k}_x^{\mathcal{L}} + \alpha)_y}{||\boldsymbol{k}_x^{\mathcal{L}} + \alpha||_1} \tag{2.15}$$

The last term is the y-th element of the normalized vector. The 1-norm is used to normalize the vector.

2.4 Risk Difference Using the Conjugate Prior

Next, we insert equation 2.15 into the empirical risk equation 2.7. We are approximating $p(y|x)$ with $p(y|\boldsymbol{k}_x^{\mathcal{L}})$ which is the empirical risk based on the labeled data $\mathcal{L}$.

$$\hat{R}_{\mathcal{E}}(f^{\mathcal{L}}, \mathcal{L}) = \frac{1}{|\mathcal{E}|} \sum_{x \in \mathcal{E}} \sum_{y \in \mathcal{Y}} \frac{(\boldsymbol{k}_x^{\mathcal{L}} + \alpha)_y}{||\boldsymbol{k}_x^{\mathcal{L}} + \alpha||_1} \cdot L\left(y, f^{\mathcal{L}}(x)\right). \tag{2.16}$$

Now lets assume we add a new labeled candidate (x_c,y_c) to the labeled data set $\mathcal{L}$. We will now denote the set with the newly labeled data point $\mathcal{L}^+ = \mathcal{L} \cup \{(x_c,y_c)\}$. Next we need to determine how much this new data point improved our classifier. We then make an estimate of the gain in terms of risk difference using the probability to estimate the ground truth.

$$\Delta\hat{R}_{\mathcal{E}}(f^{\mathcal{L}^+}, f^{\mathcal{L}}, \mathcal{L}^+) = \hat{R}_{\mathcal{E}}(f^{\mathcal{L}^+}, \mathcal{L}^+) - \hat{R}_{\mathcal{E}}(f^{\mathcal{L}}, \mathcal{L}^+) \tag{2.17}$$

$$= \frac{1}{|\mathcal{E}|} \sum_{x \in \mathcal{E}} \sum_{y \in \mathcal{Y}} \frac{(\boldsymbol{k}_x^{\mathcal{L}^+} + \alpha)_y}{||\boldsymbol{k}_x^{\mathcal{L}^+} + \alpha||_1} \cdot \Big(L(y,f^{\mathcal{L}^+}(x)) - L(y,f^{\mathcal{L}}(x))\Big) \tag{2.18}$$

The observations used to estimate the risk are the same for both the old and new classifiers. We do this because we assume that adding labeled data will make the classifier better, so this allows us to more accurately compare the current classifier and the new one.

2.5 Expected Probabilistic Gain

If we are able to reduce the error with the new $\mathcal{L}^+$ model then equation 2.18 will be negative. As a result, we negate this term and maximize the expected probabilistic gain. To simplify things we set $\alpha = \beta$.

$$\text{xgain}(x_c, \mathcal{L}, \mathcal{E}) = \mathop{\mathbb{E}}_{p(y_c|\boldsymbol{k}_{x_c}^{\mathcal{L}})} \left[-\Delta\hat{R}_{\mathcal{E}}\left(f^{\mathcal{L}^+}, f^{\mathcal{L}}, \mathcal{L}^+\right)\right] \tag{2.19}$$

$$= -\sum_{y \in \mathcal{Y}} \frac{(\boldsymbol{k}_x^{\mathcal{L}} + \beta)_y}{||\boldsymbol{k}_x^{\mathcal{L}} + \beta||_1} \cdot \frac{1}{|\mathcal{E}|} \sum_{x \in \mathcal{E}} \sum_{y \in \mathcal{Y}} \frac{(\boldsymbol{k}_x^{\mathcal{L}^+} + \alpha)_y}{||\boldsymbol{k}_x^{\mathcal{L}^+} + \alpha||_1} \cdot \Big(L(y,f^{\mathcal{L}^+}(x)) - L(y,f^{\mathcal{L}}(x))\Big) \tag{2.20}$$

Finally, for the xPAL selection strategy, we simply choose this candidate $x_c^* \in \mathcal{U}$ where the gain is maximized:

$$x_c^* = \mathop{\arg\max}_{x_c \in \mathcal{U}} \left(\text{xgain}(x_c, \mathcal{L}, \mathcal{E})\right). \tag{2.21}$$

3. Related Works

The main paper that influenced our work is called 'E-Commerce Merchant Classification using Website Information' published by Sahid et al. [2019]. In this paper the authors test different text processing strategies, embeddings, machine learning methods, and scraping methodologies.

3.1 E-commerce Merchant Classification

The authors explaining the importance of e-commerce and the need for accurate classification of e-commerce merchants for market analysis and risk management in relation to using a payment gateway service. In their case it is important to classify a merchant to see if they are in a high risk category. They then describe the proposed method, which consists of three main stages: data collection, data preprocessing, and merchant classification.

In the collection stage, the authors collect data from e-commerce websites using a web crawler. They extract the html text data from the home page and various sibling pages, depending on the experiment.

In the preprocessing stage, the authors perform several steps to clean and transform the data. They remove missing and redundant features, characters of length 1 or 2 and sequences of numbers. Next, they use TF, TF-IDF, or embedding to create their dataset.

In the merchant classification stage, the authors apply several machine learning algorithms, including Decision Trees, Naive Bayes, k-NN, MLP, Logistic Regression, and Support Vector Machine (SVM) to classify the e-commerce merchants into several categories, such as electronics, flowers, and gambling. They evaluate the performance of each algorithm using several metrics, such as accuracy, precision, recall, and F1 score.

The experimental results show that the proposed method outperforms the baseline method in terms of classification accuracy and other metrics. The SVM algorithm achieves the highest macro averaged F-score of 0.83 indicating that the proposed method is effective in classifying e-commerce merchants based on their website data.

3.2 Influence

The e-commerce paper helped guide us in our work because we used lessons they learned to tailor our approach. For example, we focused on scraping data from just the home page and using TF-IDF instead of exploring other methods. However, we also incorporate active learning which was not used in the e-commerce paper.

4. Data Review

In this chapter we take a deeper look into the data and the process of collecting, translating, and encoding. Our partner has provided a small sample of 1000 labeled data points. This data was manually labeled by a human annotator.

4.1 Overview

The provided data consists of a merchant name, merchant website (url), merchant category, and merchant tag as shown in Table 4.1.

merchant name	merchant url	merchant category	merchant tags
State Hospital	http://hospital.com/	Health	'{"Clinic"}'

Table 4.1: This is a faux example of a single data point.

The current process consists of giving the merchant url to an annotator. The annotator then views the website and can instantly (after viewing the homepage) provide a label and tags for the website. However, in some cases the annotator may need to browse further (by viewing sibling pages such as the 'About Us' section or individual product pages) to get an idea of how the website should be labeled.

The annotator simply needs to view then mentally process the text and images from the website and make some reasonable decisions on how the site should be classified. However, the annotator does not record what the content on the site said or what drove them to make their decision.As a result we are missing a key portion of data for the classification process, the text.

Tags are also used when labeling the data to provide further granularity. The merchant tags are ordered by specificity, with the first tag in the list being the most general and the final being the most specific. An example of the tag hierarchy is show in Table 4.2 where we can see that this sample consists of data from various categories all contained within the 'Eco' side tag grouping.

Category	Level 1 Tag	Level 2 Tag	Level 3 Tag	Side Tag
Travel	Local Transport	Micro-mobility	Bike Sharing	Eco
		Public Transport		Eco
Fashion	Clothing - Other	Second Hand		Eco
Car	Charging Station			Eco
	Car Sharing			Eco

Table 4.2: This is an example of how the tags use different levels.

The tags are important because they allow us to separate the data even further and group or sort the data differently. However, in our work we didn't opt to include the tags in the classification or selection strategy process at this time.

4.2 Collection

Our goal is to automate the website navigation and data collection, data storage, and classification. This has the potential to speed up the browsing process in comparison to current methods. We make the obvious checks to see if the website has already been scraped and stored in the database to ensure we are not wasting time and resources.

The labels needed to be augmented with the text from the websites. For the human annotator, this text data is simply stored in their short term memory while they view the website. Once they have a category for the website they can mostly forget about the text data and move on to assigning a category to the next website.

To gather the text data from the websites we used the Scrapy framework to extract text data from a single top level page from the website. We chose only to scrape the top level (main or home) page text because of the results published in another study where it was observed that adding more pages to the data set does not necessarily mean obtaining better classification results (Sahid et al. [2019]).

Out of these initial data points 184 contained links that could not be accessed or links that provided no text data that could be scraped. Two websites were particularly problematic. Facebook and Instagram both are used by businesses as main information webpages. However, neither site allows for simple text scraping and a more advanced approach would be needed to extract data from business with information on these platforms. In an effort to reduce the complexity of our scraper we decided to not create an additional scraper or integrate an API to handle these websites. Out of the remaining 816 data points 275 of them were in English. Out of the remaining 275 English data points the data was distributed into the categories as shown in Figure 4.1.

Our goal was to see how well the classifier and active learning sampling strategies would perform with limited data, but 275 samples with 23 categories was a bit to small and we still had a significant amount of data that wasn't being used (i.e. the untranslated data). It was clear we needed to find a way to translate the existing data. We tried various libraries available on GitHub but weren't getting good consistent results and we were hitting API request limits. After some time, we found that Azure had a service available and a free option of up to 2 million characters translated per month. This was a viable option and we were able to use this API to translate the remaining data. We limited the number of characters to 1000 per non English data point from the scraped text to avoid maxing out the API. After translating the non English data we had an additional 541 data points in English, giving us a total of 816 English samples, which was enough for us to get started with testing.

4.3 Processing

It is important for us to have the data in English as it allows us to exploit stop words when using the Scikit-Learn TF-IDF vectorizer to construct our data set. Stop words are words like "and", "the", "him", which are presumed to be uninformative in representing the content of a text, and which may be removed to avoid them being construed as signal for prediction (Pedregosa et al. [2011]).

This holds true for our data set as well because we are not analyzing the text for sentiment or other linguistic features. We are simply looking for the most common words in each category.

Before we opted to translate the data we tried to make what we had in english work. But we found that the 'Food and Drink' category has many more data points then the 'Culture' and 'Investments' categories which each had a single data point, see Figure 4.1. The 'Children' and 'Financial Services' categories weren't represented at all. Obviously this was problematic because we would like to have, minimum, three data points in each category to build train and test sets. This was the moment it was clear that our data set wasn't representing all categories equally and we needed to translate the other text.

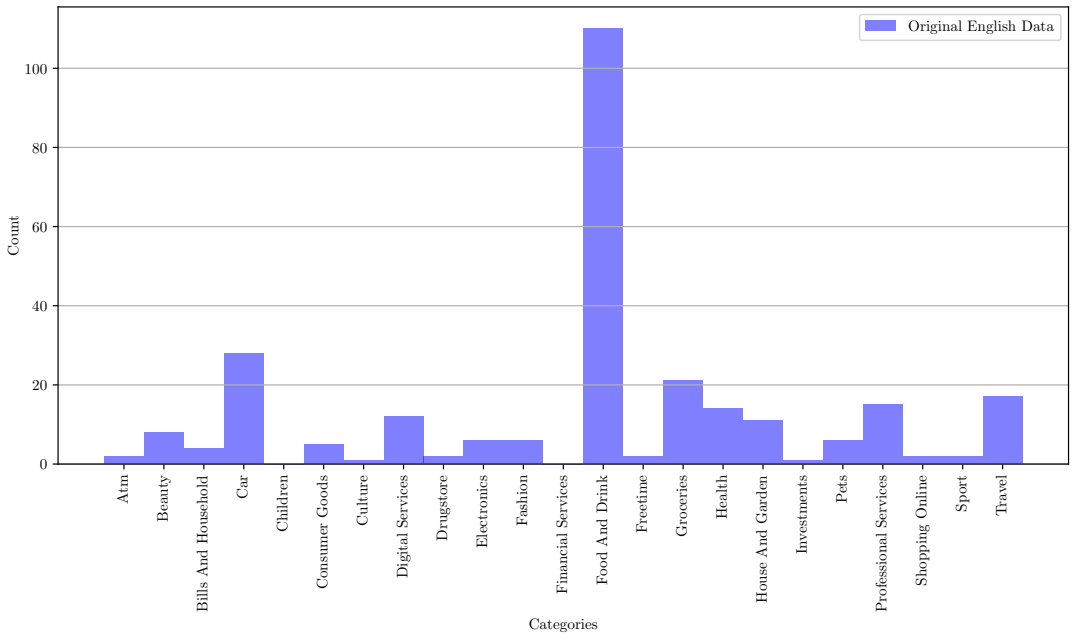


Figure 4.1: The histograms for the original usable english data.

An example of the first 100 characters of scraped text data from a website is shown in Table 4.3. The scraped text data is a single string of text that is a concatenation of all the text data pulled from the website url with the html removed.

Table 4.3: Raw text collected by scraper and the translated text.

Raw	DentalVision - Profesionální soukromá zubní klinika v centru Hradce Králové ÚvodSlužby a ceníkOrdina
Translated	DentalVision Professional private dental clinic in the center of Hradec Králové IntroductionServices

We can see that html and other symbols are removed and the majority of the words were translated. There are still some issues with words being concatenated such as 'IntroductionServices' however we do try to separate these words after translation using regex, before passing the text to the TF-IDF vectorizer.

To complement the original data we manually collected and labeled 141 additional data points for the categories that had low representation. This consisted

of searching the internet for lists of websites similar to the ones in each category. Next we would browse the site to see if it was relevant and then add it to our list of additional websites and provide it a label. This was necessary because some categories only had 2 or 3 samples, however it was quite time consuming. The additional data are almost all from English language websites, this made it easier for us to explore and provide accurate labels for the sites. These data group splits will be referenced in the following experiments and a table of the exact counts for each group can be found in the Attachments in Table A.1. In Figure 4.2. we show a bar chart of the counts of the original data and the additional data for each category and for each language, with the two letter language codes used to represent the languages.

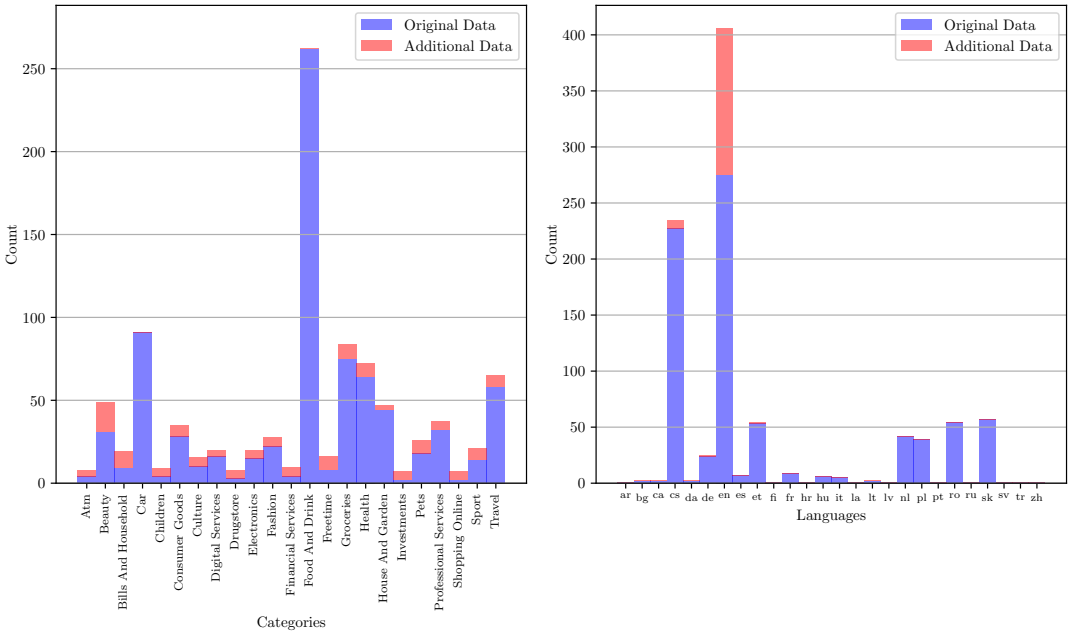


Figure 4.2: The histograms for the original and additional data for all languages.

After translating the data we used the TF-IDF vectorizer from Scikit-Learn. TF-IDF is an important tool commonly used in natural language processing and data science. The first part, TF, stands for term frequency and is a measure of how often a term appears in a document, while IDF (inverse document frequency) is a measure of how important a term is in a set of documents. The idea behind IDF is that a term that appears in many documents is less important than a term that appears in only a few documents, as the former is likely to be more common and less discriminative. The formula for calculating TF-IDF is as follows:

$$\text{TF-IDF} = TF \times IDF \tag{4.1}$$

where:

$$\text{TF} = \frac{\text{number of occurrences of term } t \text{ in document}}{\text{total number of terms in document}}$$

$$\text{IDF} = \log_e \left(\frac{\text{total number of documents}}{\text{number of documents with term } t \text{ in it}}) \right)$$

To calculate the TF-IDF score for a given term in a document, we would first calculate the term frequency then calculate the inverse document frequency multiply them to get the TF-IDF score for the term in the document as shown in Equation 4.1.

This process is repeated for each term in each document in the corpus, resulting in a TF-IDF matrix that can be used for various natural language processing tasks such as text classification, information retrieval, and clustering. It is important to understand TF-IDF because it is the basis for the feature selection we use throughout our experiments.

4.4 Statistical Tests

We explored statistical significance of the relationship between variables and identifying highly contributing variables using chi-squared analysis, variable importance, and Fisher's exact test with all useable data. We also explored dimensionality reduction techniques with PCA but found no significant results.

4.4.1 Chi-squared

Chi-squared analysis is a statistical method used to determine the association between two categorical variables. It is used to test whether two categorical variables are independent of each other or not. In other words, it helps to determine if there is a significant relationship between two variables.

The chi-squared test involves comparing the observed frequencies of each category in a contingency table to the expected frequencies. A contingency table is a two-dimensional table that shows the frequency distribution of two categorical variables. The expected frequencies are the frequencies that would be expected if the two variables were independent. The difference between the observed and expected frequencies is then squared, divided by the expected frequency, and summed over all categories to give the chi-squared statistic. The formula for calculating the chi-squared statistic is as follows:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{4.2}$$

where:

$$\begin{aligned} \chi^2 &= \text{the chi-squared statistic} \\ O_i &= \text{the observed frequency of category } i \\ E_i &= \text{the expected frequency of category } i \end{aligned}$$

The degrees of freedom for the chi-squared test are calculated as (number of rows - 1) x (number of columns - 1). The chi-squared statistic is then compared to a critical value from a chi-squared distribution table with the degrees of freedom and a specified level of significance. If the calculated chi-squared statistic is greater than the critical value, then we reject the null hypothesis that the two variables are independent.

Some categories such as 'Culture', 'Digital Services', 'Shopping Online' that have few data points have words such as 'kihnu', 'synnex', 'joom', respectively,

which have no relative meaning to the category in English but also may show the limitations of our website scraping and translation capabilities. We have a problem where some words are actually important but weren't translated correctly such as 'maso' in the 'Groceries' category, which means 'meat' in Czech. See Table 4.4 for more information.

From what we discussed in the previous section, we can see that the 'Culture' category has only one data point and the 'Digital Services' category has only two data points. This is problematic because if the single data point we have doesn't represent the category well then we will continue to have difficulty classifying until we have more robust data.

Table 4.4: Keywords from TF-IDF with chi-squared using all useable data.

	Keyword 1	Keyword 2	Keyword 3
Atm	caixa	bank	banking
Beauty	hairdressing	hairdresser	hair
Bills And Household	internet	fullness	trash
Car	parts	auto	car
Children	toy	toysrus	toys
Consumer Goods	smoke	flowers	flower
Culture	museum	kihnu	theater
Digital Services	salesforce	bitly	servers
Drugstore	walgreens	detergent	drimble
Electronics	onoff	electronics	computers
Fashion	jewellery	women	men
Financial Services	venmo	mutual	retirement
Food And Drink	bar	cafe	restaurant
Freetime	lanes	casino	bowling
Groceries	kohl	maso	bakery
Health	patient	pharmacy	dental
House And Garden	paints	garden	hardware
Investments	tesla	financial	investment
Pets	animal	veterinary	pet
Professional Services	headhunters	school	parcel
Shopping Online	default	joom	ends
Sport	hydrogen	functional	singltrek
Travel	accommodation	rooms	hotel

Realistically, chi-squared may not be the best statistical method to use with our data because we have some categories that have very few samples. However, we still wanted to also explore variable importance.

4.4.2 Variable Importance

Variable importance analysis is used to identify the most important predictors or variables that contribute to a particular outcome. This analysis can help to identify which variables are most predictive and should be included in a predictive

model, or which variables may need further investigation to better understand their relationship to the outcome.

We calculated the variable importance using the RandomForestRegressor from Scikit-Learn and provided a list of the top 20 most important words from the TF-IDF vectorizer. In the context of a random forest regression model, variable importance refers to the relative importance of each input feature used in the model. It is an ensemble machine learning technique that builds multiple decision trees, each using a random subset of the features and samples from the training data. The random forest combines the predictions of all the decision trees to generate the final output.

The variable importance in a random forest model is based on the Gini impurity. The Gini impurity measures the degree of randomness or impurity in a decision tree node. A low impurity indicates that a node contains mostly one class, while a high impurity indicates that a node contains an equal number of different classes. The Gini importance of a variable is calculated as the sum of the Gini impurity decreases across all the decision trees that used that variable as a split criterion. The higher the Gini importance of a variable, the more it contributes to reducing the overall impurity of the decision tree nodes and hence the model.

The feature_importances_ attribute returns an array of values, one for each input feature, that represent the relative importance of the feature in the random forest model, the results for our data can be found in the Attachments in Table A.2.

4.4.3 Fisher's Exact Test

Fisher's exact test is a statistical significance test used in the analysis of contingency tables. It is used to determine whether or not there is a significant association between two categorical variables. Technically, Fisher's exact test is appropriate for all sample sizes. However, the number of possible tables grows at an exponential rate. Therefore it's typically best for smaller sample sizes.

In our case, we constructed a contingency table with one category versus the rest of categories for each word. We then calculated the p-value for each word using Fisher's exact test. The p-value is the probability of observing a test statistic at least as extreme as the one that was actually observed, assuming that the null hypothesis is true. The null hypothesis is that the two variables are independent. If the p-value is less than the significance level, then we reject the null hypothesis and conclude that the two variables are dependent.

Typically a significance level of 0.05 is used but it may be wise to consider the Bonferroni correction to account for multiple comparisons in our case. As a result, we would need to adjust the significance level to 0.05 divided by the number of categories. This would give us a new significance level of 0.0022 and we would reject the null hypothesis for words with a p-value less than this. Our results can be found in the Attachments in Table A.7.

5. Testing with Original Data

Here we will explore the results of different classifiers and active learning strategies on the original data. The original data consists of data discussed previously in Figure 4.1 and shown discretely in the Attachments in Table A.1. The original data was comprised of the original English and translated data but not the additionally collected data.

5.1 PWC, RBF Kernel, and Active Learning

In our first experiment we used the radial bias function (RBF) kernel which is a popular kernel function. It is defined as:

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \tag{5.1}$$

where σ is a parameter that controls the smoothness of the kernel and x_i and x_j are the two points in the feature space to compare. The budget is the number of available samples we can use to train the classifier, where the active learning method selects the next sample to add to the labeled data. Here, it is set to 612 which is 75% of the the available samples. The remaining samples are used for calculating the test error. In later experiments we reduce the budget to decrease the evaluation time.

In Figure 5.1 we have the train and test errors for four different active learning sampling strategies. XPAL appears to find the best examples to reduce the test error the fastest but all sampling strategies appear to plateau around the 50% error mark.

5.2 PWC, Cosine Kernel, and Active Learning

In Figure 5.2 we have the train and test errors for the same four different active learning sampling strategies tested on the same data. The only change was that we used cosine kernel instead of the RBF kernel.The cosine kernel is another important kernel function that is used in many machine learning algorithms. It is defined as:

$$K(x_i, x_j) = \frac{x_i \cdot x_j}{\|x_i\| \, \|x_j\|} \tag{5.2}$$

where x_i and x_j are the two points in the feature space to compare. We found that we got better and more responsive results using the cosine kernel in comparison to the RBF kernel.

In Figure 5.2 it seemed that PAL and xPAL were able to reduce the training error, early in the training process compared to random selection and QBC. We also tested the other sampling strategies with the cosine kernel and found that the results were similar. The other sampling strategies and their test data results are shown in Figure 5.3 along with the test data from Figure 5.2.

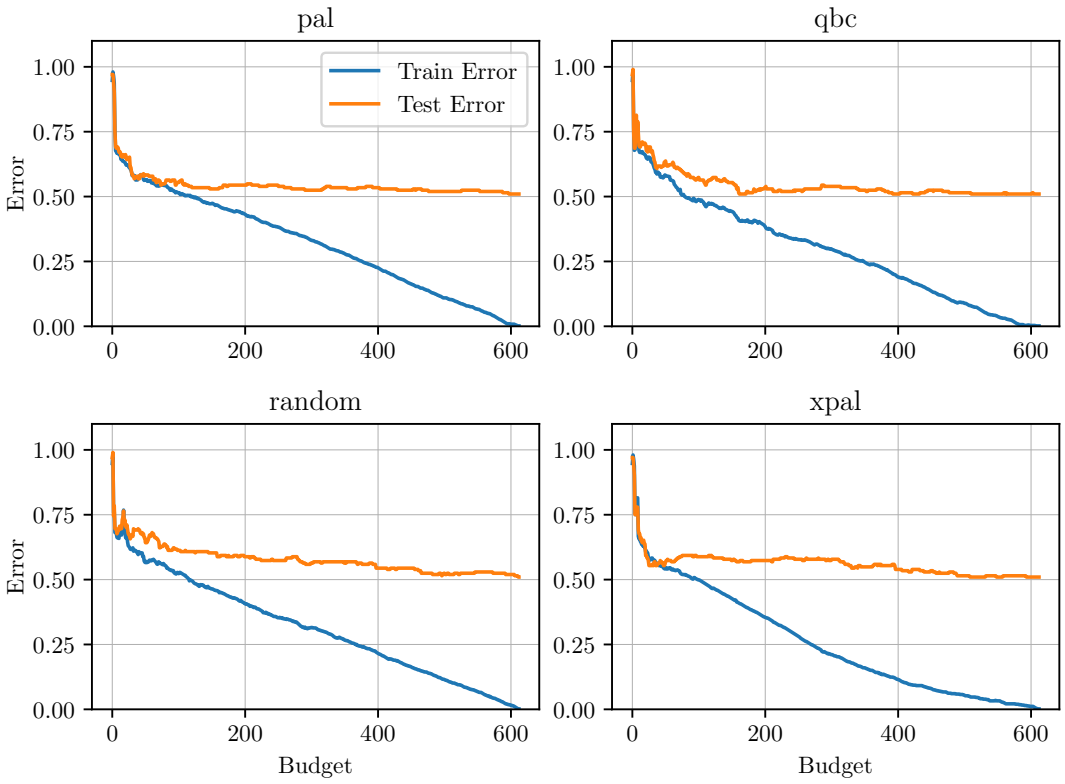


Figure 5.1: Train and test error using different query strategies and RBF kernel for the PWC classifier using the original data.

We can see that the sampling strategies test performance converges over time (as we are using the same data and classifier). XPAL appears to be performing well early on in the training process, in the 100-200 budget range. This test is only showing the results of one data split. We can get a better idea if we run more tests with different train-test splits and average the results.

Figure 5.3 shows that xPAL seems to be performing the best with our data but we wanted to see if we ran more tests with different train-test splits how the results would average out and which sampling strategy would perform the best on average. We ran 10 different data splits with each of the 7 sampling strategies and then took the average to get a smoother curve compared to the single run results shown in Figure 5.3. The results for this experiment are shown in Figure 5.4.

It is clear in Figure 5.4 that xPAL is performing the best early on (budget from 0-100) in the sample selection process. XPAL selects the data that minimizes the test error and builds the strongest classifier quickly while it takes the other sampling strategies more to get to the same level of performance. Around the 100 budget mark we can see that the other selection strategies catch up to xPAL performance wise. We want to note that for each plot where we averaged query strategies (10 runs/ splits per query strategy) that it took 24 hours on average to run the experiment on a CPU cluster, with PAL taking the longest of any of the sampling strategies.

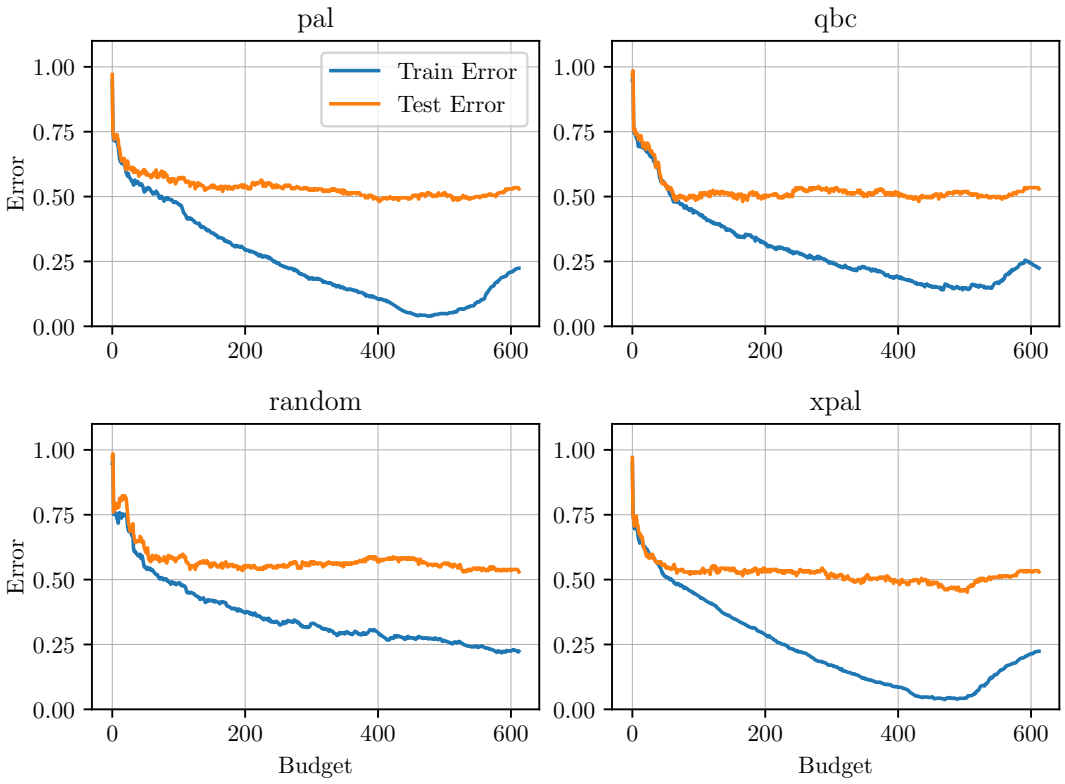


Figure 5.2: Train and test error using different query strategies and cosine kernel for the PWC classifier using the original data.

5.3 Classifier Evaluation

We also decided to test out some classifiers from the Scikit-Learn library to compare performances. We used the original data with TF-IDF vectorizer methodologies in all following chapters. It should be noted that cross validation was used here for evaluation but it was not used in the previous sections.

The goal of this exploratory phase was to try and decide which classifier to conduct more thorough testing with. As a result, we didn't use GridSearchCV for each classifier at this stage and we mostly used the default parameters and their cross validation scores with all of the original data (i.e. additional data not included). In some cases where using weights was an option for the classifier we included the precomputed cosine decay weights. A table of the parameters used for each classifier is shown in the Attachments in Table A.4.

The results for the different classifiers are shown in Figure 5.5. In the box-plot, the whiskers extend from the box to the furthest data points that are within 1.5 times the inter-quartile range (IQR) of the box. Any data points that are beyond the whiskers are considered outliers and are plotted as individual points or symbols (diamonds) as seen in the figure.

The base LinearSVC classifier model performed best compared to other classifiers and it is a fast running algorithm even with data that has many features. We decided to look further into LinearSVC because it performed well.

The LinearSVC is a type of machine learning algorithm that can be used for binary or multiclass classification problems. It's designed to predict one of two

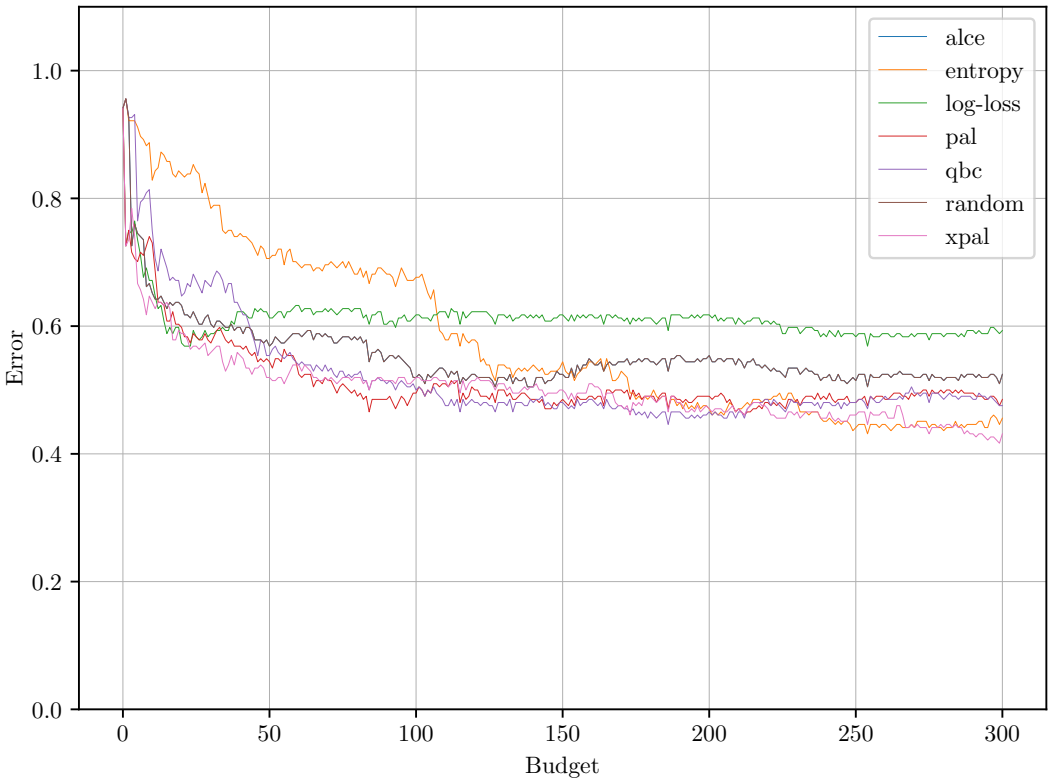


Figure 5.3: Comparing test error with one data split using different query strategies and cosine kernel for the PWC classifier using the original data.

possible outcomes based on a set of input features.

The basic idea behind the LinearSVC is to find the best line (or hyperplane) that can separate the two classes in the feature space. To do this, the algorithm looks for the line that maximizes the margin between the two classes. The margin is the distance between the decision boundary (i.e., the separating line) and the closest data points from each class. By maximizing the margin, the LinearSVC can achieve good generalization performance on new data.

In the case of multiclass classification, the LinearSVC works by dividing the data into multiple binary classification problems, one for each possible combination of classes. It then trains a separate binary classifier for each of these problems, which can then be used to classify new data points.

For example, if we have three classes A, B, C, then we have three classifiers, one for A versus B and C, one for B versus A and C, and another for C versus A and B. Once we have trained a separate LinearSVC for each binary classification problem, we can use them to classify new data points. To classify a new data point, we simply apply each binary classifier to the data point and see which classes it is predicted to belong to. If a data point is predicted to belong to more than one class, we can use a tie-breaking rule or simply choose the class with the highest predicted probability.

The LinearSVC algorithm seeks to find the hyperplane that maximally separates the classes in feature space, and weights are sometimes used that correspond to the coefficients of the hyperplane equation.

Consider another example using a binary classification problem where we have

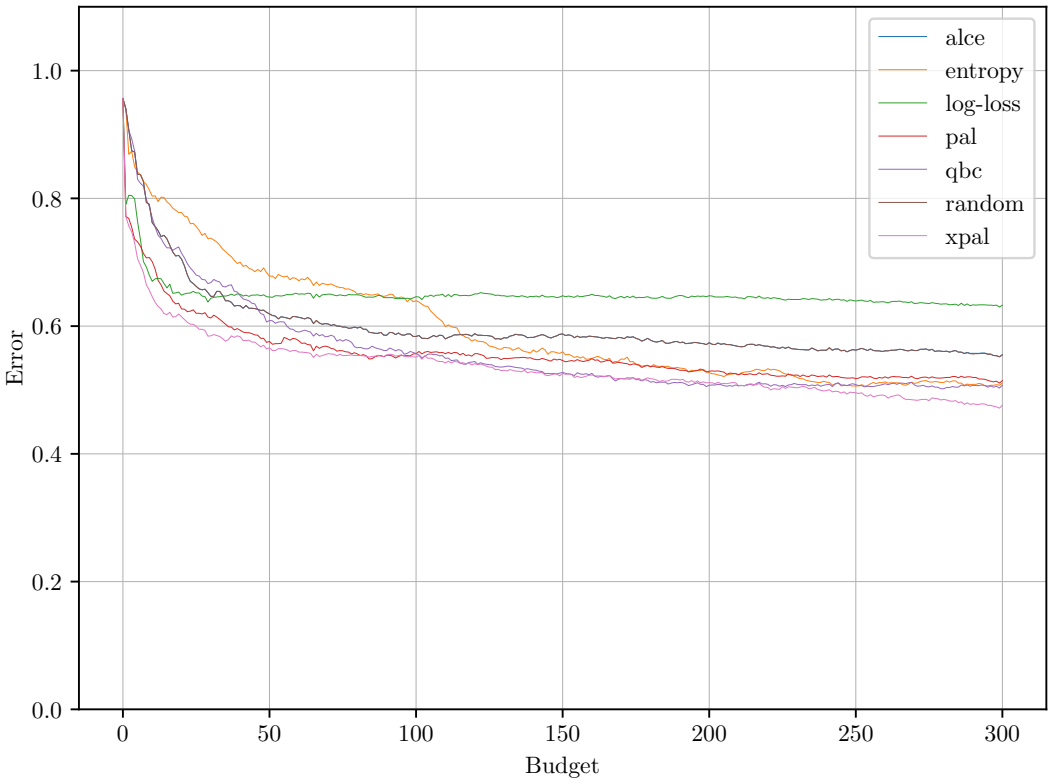


Figure 5.4: Comparing test error using different query strategies and cosine kernel for the PWC classifier with results averaged over ten different data splits using the original data.

two classes labeled as -1 and +1. Given a set of training examples, the goal of the LinearSVC algorithm is to learn a hyperplane that separates the examples of the two classes in feature space. The hyperplane is defined by the equation:

$$w^T x + b = 0 \tag{5.3}$$

where w is a weight vector of the same dimension as the feature vectors x, and b is a bias term that shifts the hyperplane in the direction of the negative class.

During training, the LinearSVC algorithm tries to find the values of w and b that minimize the classification error while also maximizing the margin between the hyperplane and the closest examples of each class. This is achieved by solving a constrained optimization problem that involves minimizing the norm of the weight vector subject to the constraint that all training examples are correctly classified with a margin of at least 1.

Once the weights are learned, they can be used to make predictions on new examples by evaluating the sign of the decision function:

$$f(x) = w^T x + b \tag{5.4}$$

If $f(x)$ is positive, the example is classified as the positive class, and if it is negative, the example is classified as the negative class. We created three different models using LinearSVC, the first was a boilerplate LinearSVC with no class

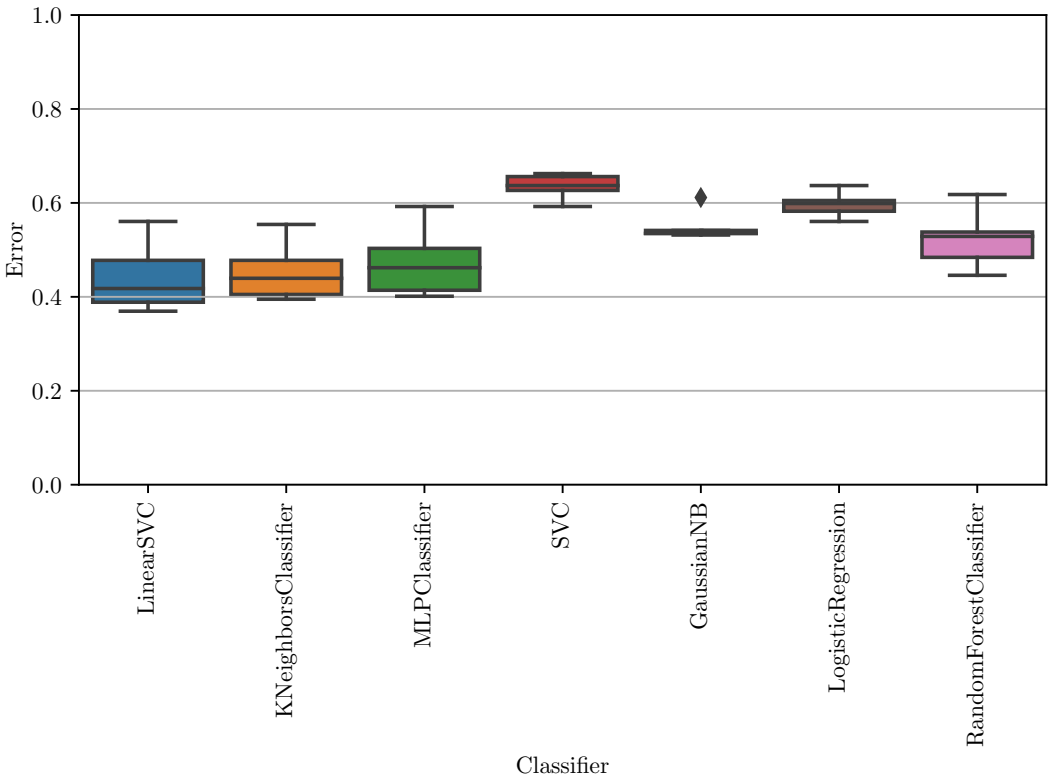


Figure 5.5: Performance of base classifiers.

weight argument modification. The second model used the class weights parameter set to 'balanced'. The 'balanced' mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as $n\_samples/(n\_classes * np.bincount(y))$. For the third test we created a dictionary of weights for each class using the cosine decay function. The weights for each category range from 0.1 to 1.0 where the most frequent classes had smaller weights. The cosine decay function is defined as:

$$w_i = \frac{1}{2}\left(1 + \cos\left(\frac{\pi t}{T}\right)\right) \tag{5.5}$$

where w_i is the weight for the i^{th} class, t is the index of the current category in the sorted list, and T is the total number of categories. The categories are counted and sorted in ascending order. The calculated cosine weights are shown in the Attachments in Table A.6.

We used the same train test spilt for these experiments and the results for are shown in Table 5.1. We also wanted to conduct more testing with KNeighborsClassifier and Neural Networks so created some small experiments to test these classifiers.

Using K-Neighbors Classifier (KNN) and a Neural Network from TensorFlow we conducted additional tests. For the KNN we found that using 8 neighbors and the cosine distance metric yielded the lowest error.

For the Neural Network we used a dense hidden-layer with 1000 neurons with Sigmoid activation and 23 output neurons with Softmax activation. For the NN optimizer we used Adamax with cosine decay with an initial learning rate of 0.1,

Table 5.1: Test error for three different LinearSVC models using the original data.

Model	Error
No Weights	0.382
Balanced Weights	0.402
Cosine Decay Weights	0.499

alpha value of 0.1, and 915 decay steps:

$$\text{Decay} = \frac{\text{Number of } X \text{ Data Samples}}{\text{Batch Size}} \cdot \text{Number of Epochs} \tag{5.6}$$

where the division is integer division. The results are shown in Table 5.2. We can see that the K-Nearest Neighbors classifier and the Neural Network classifier performed slightly worse compared to LinearSVC.

The LinearSVC outperformed the other classifiers and we decided to experiment with it further. We attempted to boost performance of the LinearSVC classifier using multiple cross validation grid searches with bagging. Bagging (bootstrap aggregating) is a type of ensemble learning, where multiple models are trained on different subsets of the training data and their predictions are combined to make the final prediction. In Scikit-Learn we used the BaggingClassifier to implement bagging. An example of our setup and parameters are shown in the code snippet in the Attachments in Figure A.1.

Performance was not improved from what we had already seen. Using bagging may not be the best approach at this stage because there are some categories that have very few samples so bagging may be unable to create a good model. We also didn't use the class weights parameter because we had already seen that it didn't improve performance. The BaggingClassifier and GridSearchCV combination didn't improve the performance of the LinearSVC classifier beyond what we had already achieved.

Table 5.2: Test error for best performing classifiers using original data.

Model	Error
LinearSVC	0.382
Tensor Flow Neural Network	0.417
K Neighbors Classifier	0.451

The precision-recall curve for the best performing classifier (LinearSVC) is shown in Figure 5.6 and the confusion matrix is shown in Figure 5.7 for the best performing LinearSVC classifier. The precision-recall curve gives us an idea at how well our classifier can correctly categorize the data. It also gives us a visualization of how unbalanced our categories are. We can see this imbalance clearly in Figure 5.6 where we have straight lines and large steps for some categories. This is a result of having a small number of data in a class. However, we can also see that for some classes the precision is relatively high even though we have few data points. Here we are namely concerned with the 'Culture' and 'Beauty' categories which have 10 and 31 data points respectively. It may be that the

keywords in the 'Culture' and 'Beauty' categories are drastically different from the other categories so the performance is better.

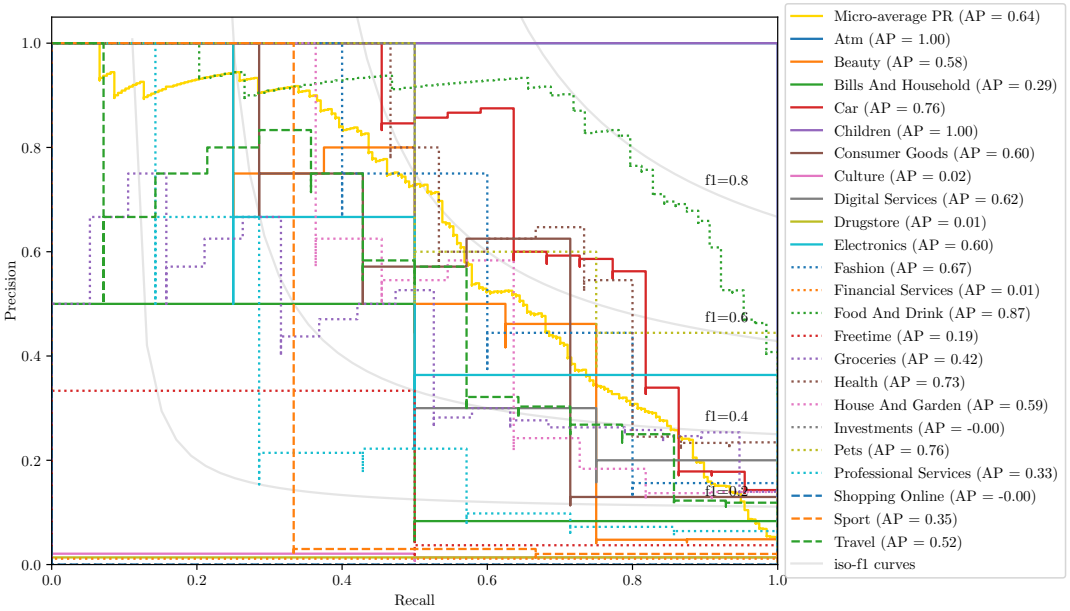


Figure 5.6: Precision-recall curve for the best performing LinearSVC using the original data.

The confusion matrix shown in Figure 5.7 may be a better metric for visualizing this data. In addition, the classification report for the LinearSVC classifier is shown in the Attachments in Table A.3 with F1, accuracy, precision, recall, and support scores.

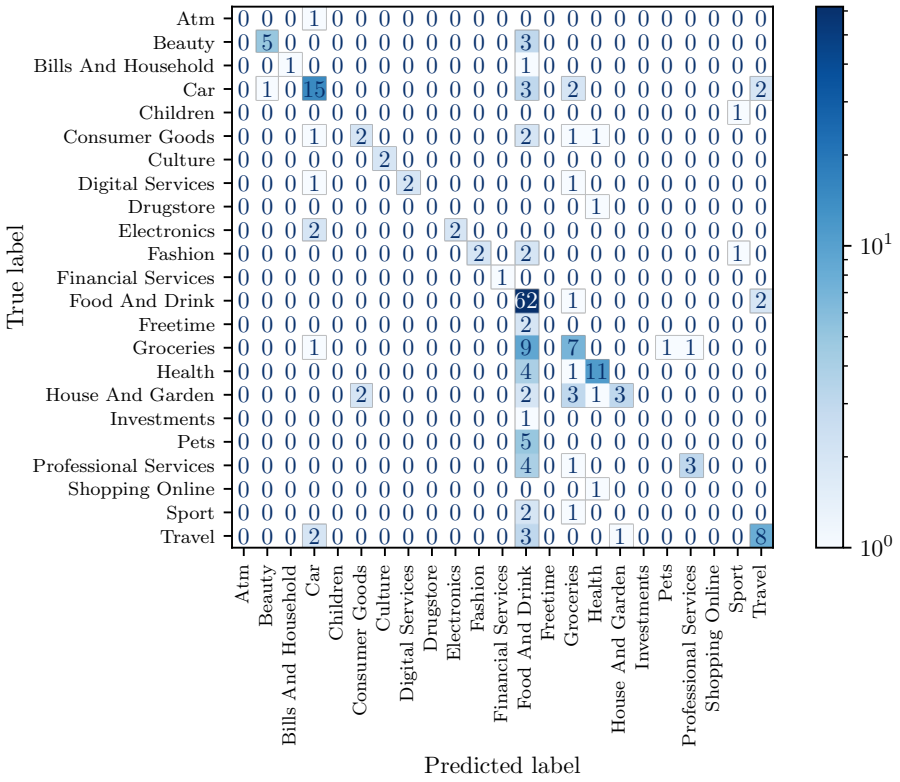


Figure 5.7: Confusion matrix for the best performing LinearSVC classifier using the original data.

6. Testing with All Data

In this section we will take what we have learned from the experiments in the previous sections and apply that knowledge to the original data set augmented with the additional data that we collected (i.e all data or all useable data). The total category count tallies can be located in the Attachments in Table A.1. We hope these experiments will provide us with a better understanding of how the data is interacting with the active learning sampling strategies and the PWC classifier. We will also try removing some data from the set if the text length is below some threshold, and evaluate the performance with a reduced number of categories.

6.1 Classifier Evaluation Revisited

We wanted to re-run the classifier experiment that we ran with the original data with the new data to get an idea of how the new data performed with the classifiers from Scikit-Learn. The results are shown in Table 6.1.

Table 6.1: Test errors for best performing classifiers using all data.

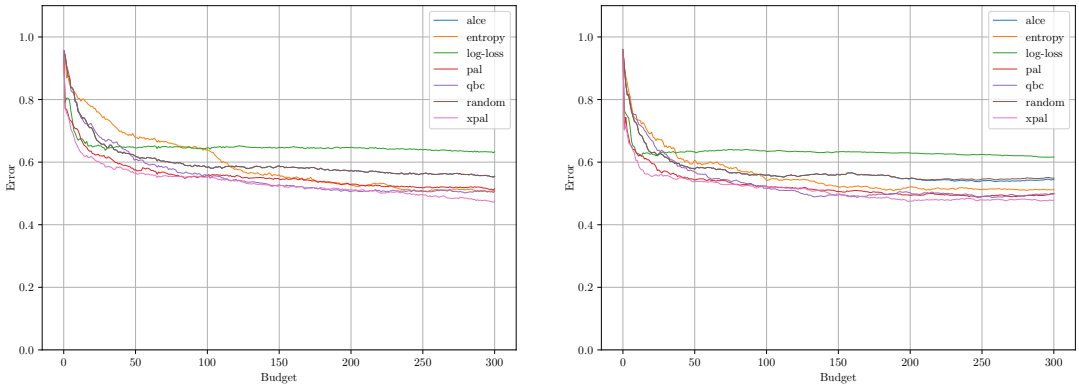
Model	Error
LinearSVC	0.429
Tensor Flow Neural Network	0.433
K Neighbors Classifier	0.471

Its interesting to see that the errors increased across the board but our previous results could have been skewed possibly because we had so few data samples in some categories. We ran the Bagging Classifier GridSearchCV from the previous chapter and again could not tune the model to perform better than the base untuned LinearSVC.

6.2 Active Learning using All Data

In these experiments we tested all the active learning methods available in the repository from the paper by Kottke et al. [2021]. We encoded all available text data using TF-IDF as we had done previously. In Figure 6.1a we show a previous experiment using just the original data so we could have a more accurate comparison with the results for all the data shown in Figure 6.2a. In Figures 6.1b and 6.2b we used an arbitrary text length filter of 50 characters to see how the performance of the selection strategies changed when we removed some of the data. The categorical data splits for the 50 character filter can be found in the Attachments in Table A.5.

We again used the different active learning methods and PWC with the cosine kernel to run our experiments. We ran 10 test runs for each method and then averaged the test error of all the runs resulting in a single curve for each sampling strategy. We only used the first 300 data points instead of using the entire budget



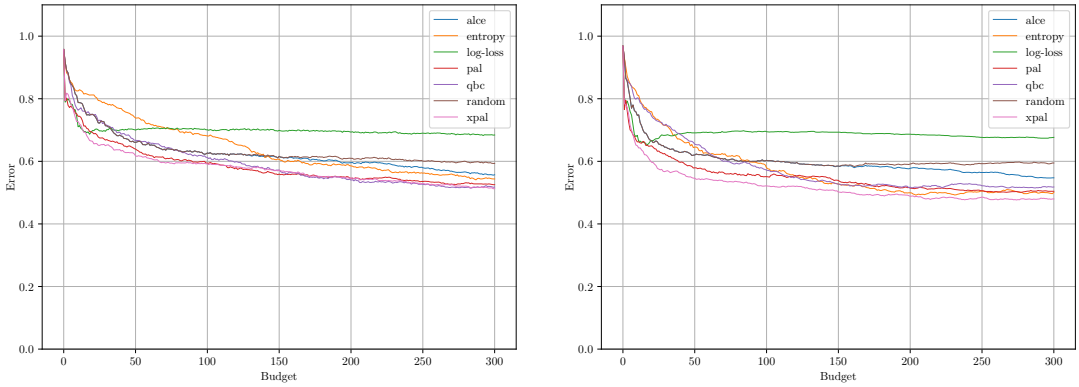
(a) No text length filter.

(b) Text length data > 50.

Figure 6.1: Active learning results using the original data.

because we want to reduce the amount of time it takes to run our experiments. As a result, the test error doesn't converge in these plots. With the original data we can see that the xPAL sampling strategy finds the best data early (20-30 budget range) in the sampling process and exploits this to improve the classifier the most. Eventually we see that PAL and QBC have similar performance to xPAL but, ultimately, after 300 samples xPAL is the best performing sampling strategy. We also want to note that in 6.1a the results for ALCE are almost entirely hidden behind the random sampling strategy curve.

Using all the data, we repeated this same experiment. The results are shown in Figure 6.2a. In both experiments in this section it appeared that the test error for xPAL was converging to around roughly 50%. But for the second experiment using all the data xPAL was finding and exploiting the best data earlier and for longer in the sampling process. We can see the dominance of xPAL throughout the plot except around the 200 budget range where entropy sampling starts to momentarily perform well.



(a) No text length filter.

(b) Text length data > 50.

Figure 6.2: Active learning results using all data.

In both the re-run original data and all data experiments, xPAL appears to perform well on average over 10 different runs in comparison to the other selection strategies, it even seemed to perform markedly well with more data available and

then having some filtering applied as shown in Figure 6.2b where it seemed to have some dominance over the other selection strategies. This led us to believe that the more data that is available and the more text that is available per data sample the better xPAL will perform. This will not always be true but it seems that if we have a large enough amount of text, 50 characters being an arbitrary example, xPAL will have an easier time selecting the best data points to label.

We were curious what xPAL was doing when more data was available, so while running the experiments we recorded when a data point was selected (its index in the selection budget) and for which category it was selected from. This data is shown in Figure 6.3.

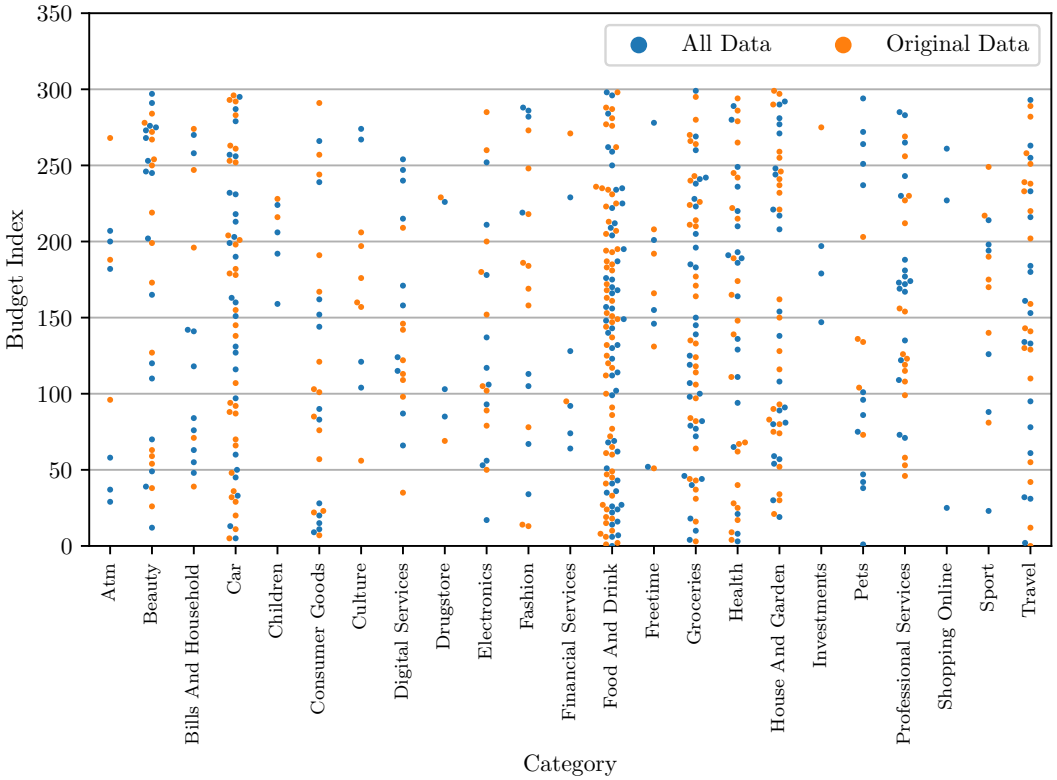


Figure 6.3: Data point selection swarm plot using xPAL.

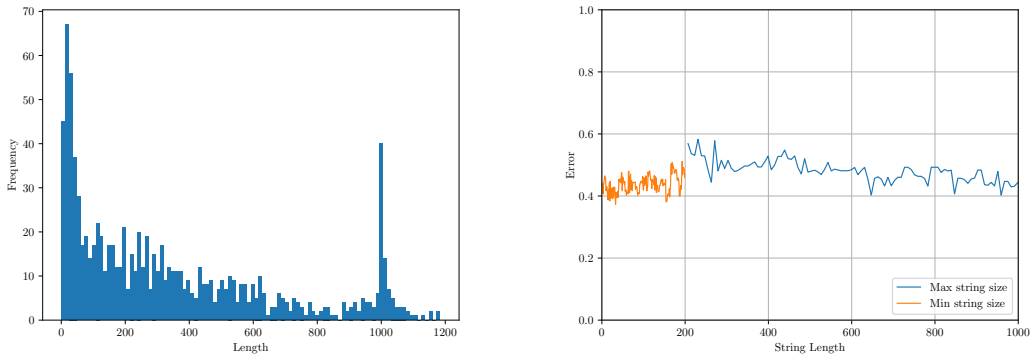
This distribution data is from a sample run for each of the data sets without any filtering. In this data split the 'Shopping Online' category for the original data went without asking for a label in the given budget window. It is interesting that it chose not to select its sole sample but, intuitively, it may make sense because there is only one other 'Shopping Online' sample point and it is in the test set. Its possible that because there is only one sample xPAL may not yet find it necessary to know its label this early in the budget and other data points labels are more valuable to know. It is clear from Figure 6.3 that when new data is available the xPAL selection strategy reevaluates what is important and selects data points that are more likely to be helpful. This is a good sign that xPAL is able to adapt to the data and find the most helpful data points.

6.3 LinearSVC with Text Length Filtering

We have some data that may not have enough text to classify correctly and others that have more than 1000 characters of text that may be noisy. We also know that sometimes while scraping the text data from a website we collected a non empty string that actually provided no words that were related to the label, such as a simple error or warning message.

Using all available data, we now have the luxury of having more than the minimum of 2 data points in some categories and can afford to remove data from the set that may have unhelpful or confusing words. We decided to explore if filtering the data based on text length could improve the performance of the classifier and use LinearSVC as it has performed well with this data previously and it is relatively fast to train.

Before conducting the filtering data experiments we can view how the text length is distributed throughout our text data as shown in Figure 6.4a. This helps us visualize how much data we may be removing if we filter the data based on text length.



(a) Distribution of text length for all data in 100 bins.

(b) Test error results for best LinearSVC with varying text length requirements.

Figure 6.4: Text length experiments data and results.

The distribution of the text show a steady decrease in length and frequency until around the 1000 length mark where we have an increase of data with large amounts of text. The spike here is a results from when we capped the non English data text to 1000 characters so that we could translate all of our data using the Azure translation services.

We imported the data and either selected a minimum number of characters or we altered the maximum number of characters allowed and created a new dataset. For each run we selected the data based on this criteria and then built the TF-IDF array. For the minimum string size tests we incremented the string size by 1 character. While for the max string size tests we decremented the string size by 10. We again used a train test split of 25% which has been our standard for testing throughout our experiments. We found that around the 200 character mark we would filter out too much data and would not have a minimum representation (2 data points) for all the categories. The results for this experiment are shown in Figure 6.4b.

We also tested with a minimum text length and a maximum text length constraint implemented together but this combination did not yield any obvious gains. This naive approach to filtering the data did not yield any obvious improvements in the test error. At best it may have filtered out some bad data and provided a small improvement in the test error. However, when implementing xPAL we hope to have it filter out the bad data in a more calculated manner.

6.4 Testing with Fewer Categories

In this section we will briefly look at the PWC with xPAL and LinearSVC performance using progressively fewer categories. We will use all available data and remove the category from the data set if it has less than the minimum number of samples. We assume that reducing the number of categories decreases the test error. Intuitively, this should be the case because as we reduce the number of categories we are reducing the number of classes that the classifier has to learn and classify. This should make the classification task easier and therefore the test error should decrease. However, we have a lot of data in the 'Food and Drink' and 'Groceries' categories and we have seen that the classifiers struggle to classify these categories correctly. Results from some tests are shown for the PWC with xPAL in Figure 6.5.

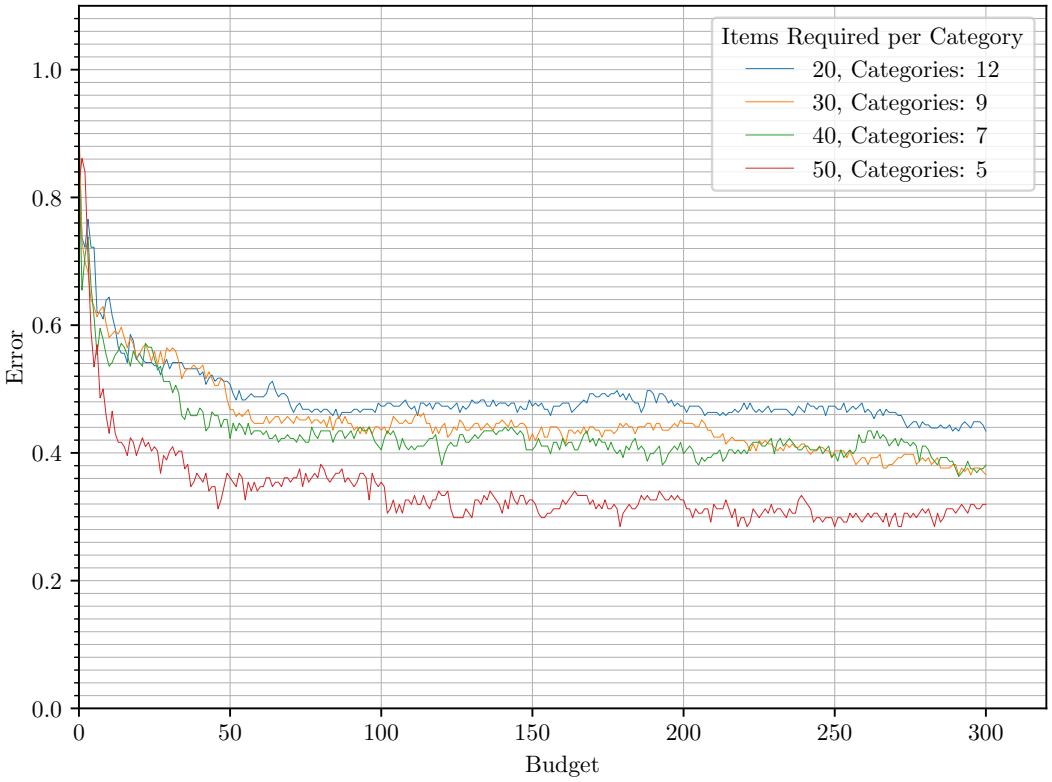


Figure 6.5: PWC with xPAL with category reduction on all data.

Dumais and Chen [2000] discuss the issue of reducing the number of categories in web content classification in their paper. They note that reducing the number of categories can improve the accuracy and efficiency of the model by reducing the

complexity of the classification problem. However, they also note that reducing the number of categories can reduce the specificity of the classification and make it more difficult to distinguish between similar categories.

Looking at the results from the xPAL and PWC tests in Figure 6.5 we can see that we have generally have reduction in test error except between the 30 and 40 category minimum tests. This table also shows us the categories that were used in the classification. This might tell us that 'Consumer Goods', 'House and Garden', and 'Professional Services' are easily linearly separable as the change in test error after 300 samples was small.

Reviewing these results it seems that 'Food and Drink' and 'Groceries' may be skewing a portion of the test error and if there was better way to classify these categories it would improve the overall test error.

Table 6.2: LinearSVC performance with category reduction on all data, where category minimum is the min number of samples required.

Category Minimum	LSVC Error	Categories	Count
20	0.359756	[Beauty, Car, Consumer Goods, Fashion, Food And Drink, Groceries, Health, House And Garden, Pets, Professional Services, Sport, Travel]	12
30	0.362416	[Beauty, Car, Consumer Goods, Food And Drink, Groceries, Health, House And Garden, Professional Services, Travel]	9
40	0.320896	[Beauty, Car, Food And Drink, Groceries, Health, House And Garden, Travel]	7
50	0.234783	[Car, Food And Drink, Groceries, Health, Travel]	5

We ran this experiment again to see if we could increase the performance by taking 'Groceries' completely out of the equation. The results are shown in Table 6.3 and the categories are the same as previously with the only difference being that 'Groceries' is not included in the test.

Table 6.3: Same as previous table but with 'Groceries' category removed.

Category Minimum	LSCV Error	Count
20	0.278912	11
30	0.303030	8
40	0.279661	6
50	0.214286	4

It is interesting to see how removing the 'Groceries' category was able to improve the test error across the board. It may be that since we have removed a category with similar data to 'Food and Drink' the classifier is able to better

classify the 'Food and Drink' category and because we have a lot of data in this category we perform better.

Table 6.4: Same as previous table but with 'Food and Drink' category removed.

Category Minimum	LSVC Error	Count
20	0.423423	11
30	0.343750	8
40	0.256098	6
50	0.190476	4

Removing the 'Food and Drink' category did not have the same effect as removing the 'Groceries' category. This may be because the 'Food and Drink' category, again, has a lot of data and removing it may have reduced the amount of data available for the classifier to learn from.

Conclusion

Our goal was to understand the entire process including the web scraping, translating, storing, and performance of the selection strategies and classifiers. This analysis provides some insight for our partner and allows them to learn from our tests and experiments.

The scraping and data collection process was an exercise in itself. We initially wanted to dockerize the entire project but this was more time consuming and cumbersome than expected, so this approach was abandoned. We setup Scrapy to take a list of websites as input and then navigate the main webpage and scrape the html. We then processed the html and saved the text locally in a Postgres database.

After the text was collected and stored we realized we had some issues because we assumed more of the text would be in English. At this stage, after inspecting the data we realized we had 9 categories with 2 or less samples and a total of 275 data points for 23 categories. However, we had a large amount of unused data that we needed to figure out how to use. We experimented with a number of API's and other tools for collecting English text but ultimately ended up using the Azure API for translation, which allowed us to have a bit larger data set of text to work with.

One issue we struggled with was the quality of the collected data. We used some statistical methods to analyze the most frequent words and sometimes found non english words as being influential for a category. This emphasized a few different things. First, was that maybe scraping text data from just a websites homepage isn't enough and experiments should be run with additional pages from the site tree. Second, that we may be lacking in quality train data as a result of our raw text scrubbing and preprocessing methods were not optimal. Finally, we could have explored better web scraping tools to collect more data from the websites. However, at a certain point the scraper was performing well enough and we decided to move on to keep with our timeline.

Our next task was to start experimenting with Scikit-Learn and TensorFlow classifiers and see how they performed on our data. We used a number of different classifiers and found that the linear support vector classifier performed well with the data. However, we only scratched the surface with TensorFlow and more testing could be done with it.

We also used the Parzen Window Classifier from the Probabilistic Active Learning GitHub repository from the "Toward optimal probabilistic active learning using a Bayesian approach" paper by Kottke et al. [2021]. This repository provided a number of different sampling strategies and we modified the repository to fit our needs and data. We found that the xPAL sampling strategy was the best for our data and was able to reduce the testing error the most compared to the other available sampling strategies in the repository based on our results.

Our work is collected into two repositories. One repository is the main repository that contains the code for the web scraper, data processing, and the thesis. The second repository is a fork of the Probabilistic Active Learning from Kottke et al. [2021] repository that contains the code for the Parzen Window Classifier, xPAL, and other sampling strategies as well as the main active learning exper-

iments, results, vectorized data from previous experiments, and text data. The raw text data scraped from the websites is stored locally. The main repository can be found at `https://github.com/borchr27/charles-university-thesis` and the Probabilistic Active Learning repository is available at `https://github.com/borchr27/probal`.

The Probabilistic Active Learning repository was setup with the PWC so that when new data was added the classifier could be updated quickly and only where there was a change or there was data effected by the new data. This caused issues as we tried to implement LinearSVC with xPAL as we would naively have to retrain the entire model for every new data point.

To improve performance we would suggest making a number of changes. One of the first changes would be to try and improve the quality of the text data from the websites. For example, a stronger web scraper could have allowed us to avoid potential IP address restriction issues and scrape data from social media websites that refused to allow our scraper to collect any data and resulted in some unusable data. We could have also run our own tests regarding how the number of sibling pages could have fortified the data.

The scraper could be upgraded to scrape social media pages. This could improve the quality of the data and allow for more accurate classification. We think that improving the quality of the data is one of the key points to improving the performance.

To be more thorough, we could have run exhaustive tests for *all* the available classifiers within Scikit-Learn using GridSearchCV and other ensemble methods to see if any more performance gains were attainable.

The current setup of the active learning xPAL process uses the PWC as it is fast (because it updates only parts of the classifier with each new data point) and relatively simple to implement. However, it is not the best method for classifying our data as we have seen from our experiments. Based on our experiments we assume that we could improve the performance of the classifier by implementing xPAL with the LinearSVC classifier, but we weren't able to test this explicitly. We discussed this with the authors of the Probabilistic Active Learning repository Kottke et al. [2021] and it could be the focus of future work.

Although there are many things that could have been improved, we still were able to increase our understanding about the data collection process, translation, storage, classification, active learning, and the nature of the data itself.

In conclusion, we found that xPAL appeared to select the best data samples to train a classifier earlier and faster than other active learning methods and we found that the LinearSVC classifier performed best in comparison to the other classifiers we tested.

Bibliography

Yoram Baram, Ran El Yaniv, and Kobi Luz. Online choice of active learning algorithms. *Journal of Machine Learning Research*, 5(Mar):255–291, 2004.

Susan Dumais and Hao Chen. Hierarchical classification of web content. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 256–263, 2000.

Yoav Freund, H Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine learning*, 28(2-3):133, 1997.

Kuan-Hao Huang and Hsuan-Tien Lin. A novel uncertainty sampling algorithm for cost-sensitive multiclass active learning. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 925–930. IEEE, 2016.

Daniel Kottke, Marek Herde, Christoph Sandrock, Denis Huseljic, Georg Krempl, and Bernhard Sick. Toward optimal probabilistic active learning using a bayesian approach. *Machine Learning*, 110(6):1199–1231, 2021.

Georg Krempl, Daniel Kottke, and Myra Spiliopoulou. Probabilistic active learning: A short proposition. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 1049–1050, Prague, Czech Republic, August 2014. IOS Press. Short Paper.

David D Lewis and William A Gale. A sequential algorithm for training text classifiers. *arXiv preprint cmp-lg/9407020*, 1994.

Robert Munro. *Human-in-the-loop machine learning*. Manning Publications, New York, NY, July 2021.

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.

Nicholas Roy and Andrew McCallum. Toward optimal active learning through monte carlo estimation of error reduction. *ICML, Williamstown*, 2:441–448, 2001.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 3rd edition, 2009. ISBN 9780136042594.

Galuh Tunggadewi Sahid, Rahmad Mahendra, and Indra Budi. E-commerce merchant classification using website information. In *Proceedings of the 9th International Conference on Web Intelligence, Mining and Semantics*, pages 1–10, 2019.

H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, 1992.

List of Figures

4.1 The histograms for the original usable english data. 19
4.2 The histograms for the original and additional data for all languages. 20

5.1 Train and test error using different query strategies and RBF kernel for the PWC classifier using the original data. 25
5.2 Train and test error using different query strategies and cosine kernel for the PWC classifier using the original data. 26
5.3 Comparing test error with one data split using different query strategies and cosine kernel for the PWC classifier using the original data. 27
5.4 Comparing test error using different query strategies and cosine kernel for the PWC classifier with results averaged over ten different data splits using the original data. 28
5.5 Performance of base classifiers. 29
5.6 Precision-recall curve for the best performing LinearSVC using the original data. 31
5.7 Confusion matrix for the best performing LinearSVC classifier using the original data. 32

6.1 Active learning results using the original data. 34
6.2 Active learning results using all data. 34
6.3 Data point selection swarm plot using xPAL. 35
6.4 Text length experiments data and results. 36
6.5 PWC with xPAL with category reduction on all data. 37

A.1 Bagging classifier setup. 53

List of Tables

2.1 Variable names and descriptions. 12

4.1 This is a faux example of a single data point. 17
4.2 This is an example of how the tags use different levels. 17
4.3 Raw text collected by scraper and the translated text. 19
4.4 Keywords from TF-IDF with chi-squared using all useable data. . 22

5.1 Test error for three different LinearSVC models using the original data. 30
5.2 Test error for best performing classifiers using original data. . . . 30

6.1 Test errors for best performing classifiers using all data. 33
6.2 LinearSVC performance with category reduction on all data, where category minimum is the min number of samples required. 38
6.3 Same as previous table but with 'Groceries' category removed. . . 38
6.4 Same as previous table but with 'Food and Drink' category removed. 39

A.1 Category counts of data. Orig. = Orig. English + Translated, is the data from the original set that had usable text. Orig. English is a subset of Orig. that was in English. Translated is a subset of Orig. that was not in English. Add. is the newly collected data, translated if needed. All Useable = Orig. English + Translated + Add. 46
A.2 Variable importance, top 20 words using all data. 47
A.3 Best LinearSVC model classification report using original data. . . 48
A.4 Set of parameters used to test all other Scikit-Learn classifier shown in Figure 5.5. 49
A.5 Data counts with 50 string character filter minimum. 50
A.6 Cosine decay weights for each category. 51
A.7 Fisher's exact tests for each category with all useable data. 52

A. Attachments

Table A.1: Category counts of data. Orig. = Orig. English + Translated, is the data from the original set that had usable text. Orig. English is a subset of Orig. that was in English. Translated is a subset of Orig. that was not in English. Add. is the newly collected data, translated if needed. All Useable = Orig. English + Translated + Add.

Category	Orig.	Orig. English	Translated	Add.	All Useable
Atm	4	2	2	4	8
Beauty	31	8	23	18	49
Bills And Household	9	4	5	10	19
Car	91	28	63	0	91
Children	4	0	4	5	9
Consumer Goods	28	5	23	7	35
Culture	10	1	9	6	16
Digital Services	16	12	4	4	20
Drugstore	3	2	1	5	8
Electronics	15	6	9	5	20
Fashion	22	6	16	6	28
Financial Services	4	0	4	6	10
Food And Drink	262	110	152	0	262
Freetime	8	2	6	8	16
Groceries	75	21	54	9	84
Health	64	14	50	8	72
House And Garden	44	11	33	3	47
Investments	2	1	1	5	7
Pets	18	6	12	8	26
Professional Services	32	15	17	5	37
Shopping Online	2	2	0	5	7
Sport	14	2	12	7	21
Travel	58	17	41	7	65
TOTALS	816	275	541	141	957

Table A.2: Variable importance, top 20 words using all data.

	Importance
hotel	0.078177
hair	0.046888
car	0.045024
auto	0.017192
services	0.014782
station	0.013815
flower	0.010145
spa	0.010054
search	0.009680
energy	0.009531
internet	0.009404
parking	0.009226
barber	0.007874
service	0.007823
stations	0.007491
beauty	0.007353
hairdresser	0.007080
rental	0.006944
banking	0.006849
mobile	0.006536

Table A.3: Best LinearSVC model classification report using original data.

	precision	recall	f1-score	support
Atm	0.00	0.00	0.00	1.00
Beauty	0.75	0.38	0.50	8.00
Bills And Household	0.00	0.00	0.00	2.00
Car	0.63	0.55	0.59	22.00
Children	0.00	0.00	0.00	1.00
Consumer Goods	1.00	0.29	0.44	7.00
Culture	0.00	0.00	0.00	2.00
Digital Services	1.00	0.50	0.67	4.00
Drugstore	0.00	0.00	0.00	1.00
Electronics	0.50	0.25	0.33	4.00
Fashion	1.00	0.40	0.57	5.00
Financial Services	0.00	0.00	0.00	1.00
Food And Drink	0.49	0.94	0.64	64.00
Freetime	0.00	0.00	0.00	2.00
Groceries	0.50	0.21	0.30	19.00
Health	0.73	0.53	0.62	15.00
House And Garden	0.50	0.45	0.48	11.00
Investments	0.00	0.00	0.00	0.00
Pets	1.00	0.50	0.67	4.00
Professional Services	0.50	0.14	0.22	7.00
Shopping Online	0.00	0.00	0.00	0.00
Sport	0.00	0.00	0.00	3.00
Travel	0.70	0.50	0.58	14.00
micro avg	0.55	0.55	0.55	197.00
macro avg	0.40	0.24	0.29	197.00
weighted avg	0.57	0.55	0.51	197.00

Table A.4: Set of parameters used to test all other Scikit-Learn classifier shown in Figure 5.5.

Classifier	Parameters
LinearSVC	{'C': 1.0, 'class_weight': None, 'dual': True, 'fit_intercept': True, 'intercept_scaling': 1, 'loss': 'squared_hinge', 'max_iter': 10000, 'multi_class': 'ovr', 'penalty': 'l2', 'random_state': 42, 'tol': 0.0001, 'verbose': 0}
KNeighborsClassifier	{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'cosine', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 8, 'p': 2, 'weights': 'uniform'}
MLPClassifier	{'activation': 'relu', 'alpha': 0.0001, 'batch_size': 'auto', 'beta_1': 0.9, 'beta_2': 0.999, 'early_stopping': False, 'epsilon': 1e-08, 'hidden_layer_sizes': 500, 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'max_fun': 15000, 'max_iter': 100, 'momentum': 0.9, 'n_iter_no_change': 10, 'nesterovs_momentum': True, 'power_t': 0.5, 'random_state': 42, 'shuffle': True, 'solver': 'adam', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': False, 'warm_start': False}
SVC	{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False}
GaussianNB	{'priors': None, 'var_smoothing': 1e-09}
LogisticRegression	{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_ratio': None, 'max_iter': 1000, 'multi_class': 'auto', 'n_jobs': None, 'penalty': 'l2', 'random_state': 42, 'solver': 'lbfgs', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}
RandomForestClassifier	{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 200, 'n_jobs': None, 'oob_score': False, 'random_state': 42, 'verbose': 0, 'warm_start': False}

Table A.5: Data counts with 50 string character filter minimum.

	All w Str Filter	Original w StrFilter
Atm	8	4
Beauty	49	31
Bills And Household	19	9
Car	91	91
Children	9	4
Consumer Goods	35	28
Culture	16	10
Digital Services	20	16
Drugstore	8	3
Electronics	20	15
Fashion	28	22
Financial Services	10	4
Food And Drink	262	262
Freetime	16	8
Groceries	84	75
Health	72	64
House And Garden	47	44
Investments	7	2
Pets	26	18
Professional Services	37	32
Shopping Online	7	2
Sport	21	14
Travel	65	58

Table A.6: Cosine decay weights for each category.

	Weight
Shopping Online	1.000
Investments	0.996
Drugstore	0.983
Atm	0.963
Financial Services	0.934
Children	0.899
Freetime	0.857
Bills And Household	0.810
Culture	0.757
Sport	0.701
Electronics	0.642
Digital Services	0.581
Pets	0.519
Fashion	0.458
Consumer Goods	0.399
Beauty	0.343
Professional Services	0.290
House And Garden	0.243
Travel	0.201
Health	0.166
Groceries	0.137
Car	0.117
Food And Drink	0.104

Table A.7: Fisher's exact tests for each category with all useable data.

category	p-value	keyword
Atm	0.000136	investing
Atm	0.003265	territories
Atm	0.007307	advisor
Beauty	0.019653	browser
Beauty	0.026494	plus
Beauty	0.057132	canada
Bills And Household	0.107499	documents
Bills And Household	0.145065	internet
Car	0.000001	car
Car	0.004088	customer
Car	0.004399	global
Consumer Goods	0.152154	selection
Digital Services	0.009286	solutions
Digital Services	0.030111	start
Digital Services	0.043597	mobile
Electronics	0.010449	electronic
Financial Services	0.000112	debit
Financial Services	0.000112	accounts
Food And Drink	0.000008	events
Food And Drink	0.000136	location
Food And Drink	0.000410	english
Groceries	0.015781	store
Groceries	0.021125	products
Groceries	0.041137	save
Health	0.000175	procedure
Health	0.000319	resources
Health	0.000980	schedule
Investments	0.053213	investment
Investments	0.064524	financial
Professional Services	0.000009	business
Professional Services	0.000257	media
Professional Services	0.000321	reports
Travel	0.000007	check
Travel	0.000488	transfer
Travel	0.000529	rate

Figure A.1: Bagging classifier setup.

```
intercepts = np.linspace(0.1, 1, 20)
c_vals = np.linspace(0.1, 1000, 20)
base_classifier = LinearSVC(
    random_state=args.seed,
    max_iter=10000)
bagging_classifier = BaggingClassifier(
    base_estimator=base_classifier,
    n_estimators=10,
    random_state=args.seed)
params = {
    'base_estimator__random_state': [args.seed],
    'base_estimator__intercept_scaling':
        np.concatenate((intercepts, [1])),
    'base_estimator__loss': ['hinge',
        'squared_hinge'],
    'base_estimator__penalty': ['l2'],
    'base_estimator__C':
        np.concatenate((c_vals, [1])),
    'base_estimator__multi_class':
        ['ovr']
    }
grid_search = GridSearchCV(
    estimator=bagging_classifier,
    param_grid=params,
    cv=5,
    scoring='accuracy',
    n_jobs=6)
```