HIERARCHICAL ACTIVE LEARNING APPLICATION TO MITOCHONDRIAL DISEASE PROTEIN DATASET

by

James D. Duin

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Stephen Scott

Lincoln, Nebraska

March, 2017

HIERARCHICAL ACTIVE LEARNING APPLICATION TO MITOCHONDRIAL DISEASE PROTEIN DATASET

James D. Duin, M.S.

University of Nebraska, 2017

Adviser: Stephen Scott

This study investigates an application of active learning to a protein dataset developed to identify the source of mutations which give rise to mitochondrial disease. The dataset is labeled according to the protein's location of origin in the cell; whether in the mitochondria or not, or a specific target location in the mitochondria's outer or inner matrix or its ribosomes. This dataset forms a labeling hierarchy. A new approach is investigated to learn the high-level classifier, i.e. whether the protein is a mitochondrion, by separately learning finer-grained target compartment concepts and combining the results. This approach is termed *active over-labeling*. In experiments on the protein dataset it is shown that active over-labeling improves area under the precision-recall curve compared to standard passive or active learning. Finally, because finer-grained labels are more costly to obtain, alternative strategies exploring using fixed proportions of a given budget to buy fine vs coarse labels at various costs are compared and presented. An approach robust to differences in label cost and budget using a multi-armed bandit to dynamically choose the label granularity to purchase is also discussed.

DEDICATION

This thesis is dedicated to my parents Paul and Vicki Duin and fiancée Anna Spady.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Stephen Scott for continual guidance in investigating this research topic. Yugi Mo and Dr. Douglas Downey for their work in developing the HAL and BANDIT methodologies. I would like to thank Dr. Juan Cui, Jiang Shu, Kevin Chiang for their assistance accessing and understanding the protein dataset that is the subject of the paper.

Contents

Chapter 1

Introduction

This study investigates an application of the Support Vector Machine and Logistic Regression machine learning algorithms to a protein dataset labeled according to a protein's location of origin in a cell. The task of classifying a given protein's location of origin can be essential in identifying the source of a mutation and helpful in the treatment of various mitochondrial diseases [1]. The dataset is labeled according to a hierarchical scheme or labeling tree, at the root level is whether the protein originates from the mitochondria or not, then the hierarchy breaks down further into specific target compartments at the leaf nodes.

Our investigation shows that leveraging separate fine-grained classifiers for each of the target compartments produces a higher performing classifier at the highest level in the hierarchy. Furthermore, a new approach in the active learning setting termed *active over-labeling* is applied to this dataset. The approach uses a certain proportion of the purchase budget to solicit labels at a finer level of granularity than the target concept. Purchasing fine-grained labels in each round of active learning produces a higher performing root-level (coarse) classifier than purchasing coarse labels alone. Analysis for this dataset is performed showing that the active approach of selecting the most uncertain labels

significantly outperforms the passive approach of selecting labels at random. The fine-grained labels also incur a higher cost than coarse-grained labels for this dataset, so multiple cost ratios are investigated and an optimal fixed fine ratio (FFR) purchasing strategy is determined for each fine cost. An approach optimally selecting FFR strategies throughout the rounds using a multi-armed strategy is also presented and existing experiments using this approach discussed in section 2.1.

1.1 Machine Learning

Machine learning (ML) algorithms are defined as computer programs that learn from experience E with respect to some class of tasks T and performance measure P, if their performance at tasks in T, as measured by P, improves with experience E [2]. In the context of this paper, the machine learning algorithms that are used include Support Vector Machines (SVM), and Logistic Regression (Logit). This work uses implementations by the sci-kit learn python library [3] for both algorithms. Our experiment requires a binary classification task, each algorithm takes a protein instance with a list of 449 features as an input and then outputs a 0 or 1 whether or not the protein belongs to a class. A separate classifier is trained for each class in the protein dataset. Both Logit and SVM classifiers have a decision function method that outputs the predicted confidence score for a given sample, which is the signed distance of that sample to the learned hyperplane.

Logit is a linear model for classification. The classifier function is shown in *eqn. 1.1*. Where x is the vector of features, theta is vector of learned parameters, the function $g(z)$ is the sigmoid function [4].

$$\begin{aligned} h_\theta(x) &= g(\theta \cdot x) \\ g(z) &= \frac{1}{1+e^{-z}} \end{aligned} \tag{1.1}$$

The theta parameters are solved for in order to minimize the sum of square errors (L2-norm) in the training set, and also to regularize the theta parameters to prevent them from getting to large and overfitting the dataset [5]. A cost function is used to solve for theta, this function is shown in *eqn 1.2*. The C parameter is the inverse of the regularization strength, a larger value means a stronger regularization [5].

$$min_{\theta,c}\frac{1}{2}\theta^T\theta + C\sum_{n}^{i=1} log(exp(-y_i(X_i^T\theta + c)) + 1) \tag{1.2}$$

A SVM constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which is used to output a classification for a given instance. The goal is to learn a hyper-plane that has the largest distance between training data points of separate classes, this is called functional margin [5]. In general the larger the functional margin the lower the generalized error of the classifier. SVMs are a maximum functional margin method that allow the model to be written as a sum of the influence of a subset of the training instances [6]. This output is given by kernel functions that are measures of similarity between data instances. The SVM implementation used solves the *eqn. 1.3*, where e is a vector of all ones, C is the penalty parameter of the error term.

$$min_{\alpha}\frac{1}{2}\alpha^T Q\alpha - e^T\alpha,\ subject\ to\ y^T\alpha = 0\ where\ 0 \leq \alpha_i \leq C, i = 1, ..., n \tag{1.3}$$

The Q function is defined in *eqn. 1.4*, where K is the kernel function.

$$Q_{ij} \equiv y_i y_j K(x_i, x_j),\ where\ K(x_i, x_j) = \phi(x_i)^T\phi(x_j) \tag{1.4}$$

The C parameter trades off misclassification of training examples against simplicity of the decision surface [5]. A low C makes the decision surface smooth, while a high C aims at classifying all the training examples correctly by allowing the model to select

more samples as support vectors [5]. The gamma (γ) parameter defines the significance a single training example can have, with low values corresponding to a single instance having a large significance to the learned hyperplane [5]. The gamma parameter can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.

In this work the following kernel functions were tested [5]:

- Linear: $\left\langle x, x^{'} \right\rangle$
- Polynomial: $(\gamma \left\langle x, x^{'} \right\rangle + r)^{d}$. d is the degree of the polynomial, and r is a coefficient passed to the solver, default is 0.
- Radial Basis Function (RBF): $(-\gamma|x - x^{'}|^{2})$. γ is the kernel coefficient.
- Sigmoid: $sigmoid(tanh(\gamma \left\langle x, x^{'} \right\rangle + r))$ r is a coefficient passed to the solver, default is 0.

1.2 Evaluating Classifier Performance

The classifier performance is primarily evaluated using Precision-Recall (PR) and Receiver Operator Characteristic (ROC) curves, although accuracy, precision, recall, confusion matrix and F-measure are also calculated. Accuracy is the total number of correctly classified instances in the test set divided by the total number of instances in the test set. The confusion matrix outputs a 2 by 2 matrix for the binary classification task. Each cell ($C_{row,col}$) in the matrix corresponds to one of the following metrics:

- True-Negatives (T_n): location $C_{0,0}$, correctly classified negative instances.
- False-Negatives (F_n): location $C_{0,1}$, incorrectly classified negative instances.
- False-Positives (F_p): location $C_{1,0}$, incorrectly classified positive instances.

- True-Positives (T_p): location $C_{1,1}$, correctly classified positive instances.

Precision is a measure of result relevancy and is given in *eqn. 1.5*.

$$P = \frac{T_p}{T_p + F_p} \tag{1.5}$$

Recall is a measure of how many truly relevant results are returned and is given in *eqn. 1.6*.

$$R = \frac{T_p}{T_p + F_n} \tag{1.6}$$

The F-measure or F1-measure (f1) is the harmonic mean of precision and recall and is given in *eqn. 1.7*.

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{1.7}$$

PR curves are constructed first by outputting the decision function score for each instance in the test set. Each score defines a threshold for computing precision, recall and f1 metrics. Precision and Recall are computed for each threshold and plotted on the PR curve, with recall on the x-axis and precision on the y-axis. ROC curves also evaluate classifier output quality. ROC curves are constructed similar to the PR curve except F_p replaces recall on the x-axis and T_p replaces precision on the y-axis. For both curves larger area under the curve (AUC) is usually correlated to a higher performing classifier.

1.3 Hierarchical Bioinformatics Data Set

Bioinformatics is a field using computer science tools and techniques for solving problems in molecular biology. The focus of this work is a Bioinformatics dataset developed in

order to identify the source of a certain class of mutations causing mitochondrial disease. Mitochondria are present in every cell of the body, with the exception of red blood cells [1]. Mitochondrial diseases may be caused by mutations in the proteins that reside within the mitochondria. These mutations that occur in locally transcribed and translated mitochondrial DNA (mtDNA), or in nuclear DNA (nDNA) whose protein products are imported into the mitochondria. These nDNA have many target locations including the mitochondria's outer and inner membrane, its matrix, and its ribosomes. Identifying the source of the mutation is an important problem in the treatment of a mitochondrial disease. It is an essential classification task to determine wether or not the offending mutation occurs in the mitochondrion or in an imported protein [1]. The positive dataset is composed of 962 human mitochondrial proteins from the Mitoproteome dataset [7]. The negative dataset is composed of 19,136 experimentally validated human proteins from UniProt [8]. A total of 1099 features were assembled by Kevin Chiang from Dr. Cui's bioinformatics lab at University of Nebraska at Lincoln (UNL), these features are described along with references to their sources in *Table* **??**. The features were reduced and combined into a resulting set of 449 dimensions [1]. This bioinformatics dataset is used for experimentation throughout this work.

Table 1.1: Features of the protein dataset along with their respective sources.

Type of Properties	Features (dimension)	Sources
General sequence features	Amino acid composition (20), sequence length (1), di-peptides composition (400)	Calculated by Kevin Chiang at UNL [1]
	Normalized Moreau-Broto, autocorrelation (240), Moran autocorrelation (240), Geary autocorrelation (240), Sequence order (160), Pseudo amino acid composition (50)	Profeat [9]
Physico chemical properties	Hydrophobicity (21), normalized Van der Waals volume (21), polarity (21), polarizability (21), charge (21), secondary structure (21) and solvent accessibility (21)	Computed with three descriptors: composition (C), transition (T), and distribution (D) [10]
	Solubility (1), unfold-ability (1), disorder regions (3), global charge (1) and hydrophobility (1)	PROSO [11], Phobius [12]
Structural properties	Secondary structural content (4), shape (Radius Gyration) (1)	SSCP [13]
Domains and motifs	Signal peptide (1), transmembrane domains (alpha helix and beta barrel) (5), Glycosylation (both N-linked and O-linked) (4), Twin-arginine signal peptides motif (TAT) (1)	SignalP [14], TMB-Hunt [15], NetOgly [16], TatP [17]

The dataset composes a classification problem, each protein is labeled according to where it originates in the cell. At the root is is whether or not the protein resides within the mitochondria, then there are the sub level labels if the protein has a separate target compartment specifications. The complete tree along with the number of instances belonging to the each label is included in *Figure 1.1*.

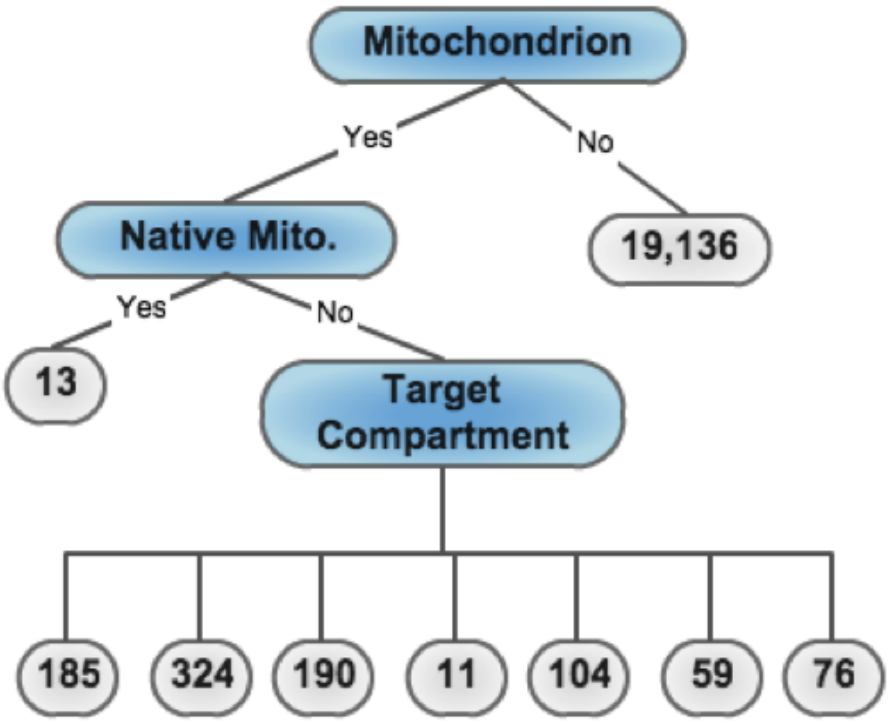


Figure 1.1: The protein dataset hierarchy of labels along with the instance count for each label.

1.4 Coarse-grained vs Fine-grained Trade Off

In hierarchically labeled datasets, over-labeling refers to learning fine-grained (non-root) concepts and combining the results to predict the coarse-grained (root) label [18]. It can be demonstrated through a simple example that for certain datasets, a fine-grained approach to the root level classifier can achieve higher levels of precision for the same level of recall. Such a dataset is shown in *Figure 1.2*. The classifiers for this dataset can be thought of as a function of axis parallel rectangular boxes. For the coarse-grained to have high recall and return all of the positive circle instances, it must encompass the entire dataset and incidentally return all of the negative diamond instances as positive also. A fine-grained approach is preferable for the simple dataset pictured. The fine-grained

classification approach for a root level classifier will achieve higher levels of precision for the same level of recall when applied to the protein dataset. In the simple example, in order for the coarse-grained learner to have high recall, precision must be sacrificed for a large amount of false positives returned. By combining fine-grained classifiers, the same level of recall can be achieved with a higher level of precision because none of the false positive diamonds will be returned.

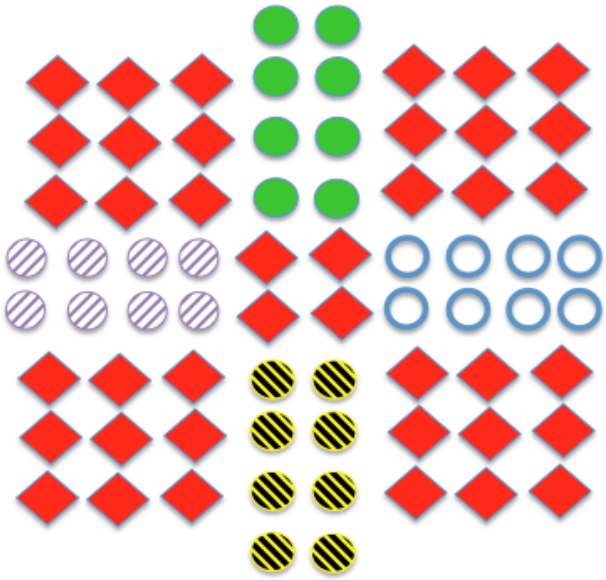


Figure 1.2: Demonstration of a dataset that would benefit from multiple fine-grained learners for each circle type.

Chapter 2

Background and Theory

2.1 Active Over-Labeling

Active Learning is when the learner is able to generate instances x itself and ask a supervisor to provide the corresponding r value during learning one by one, instead of passively being given a training set [2]. In conventional active learning the task is to learn a target concept $f : \mathcal{X} \rightarrow \mathcal{Y} = \{0, 1\}$, where $\mathcal{X}$ is the input space. The initial state is a pool of unlabeled examples $U \subset \mathcal{X}$. At each iteration, an oracle is queried at some cost for the label of an instance $u \in U$, then L is training on the labeled examples (x, y), with the goal of outputting a relatively high performing classifier for a low cost.

Since our dataset is oracle labeled, this work extends the conventional active learning approach to solicit labels at finer levels of the hierarchy for a specified cost. The oracle in this setting returns a vector of labels, corresponding to the path starting at the root of the hierarchy tree or labeling tree. An example labeling tree for the Reuters Corpus Volume I (RCV1) dataset used in text categorization research [19]. A labeling tree for this dataset is shown in is shown in *Figure 2.1*. An instance in this dataset could be labeled as ⟨Location,Building,Museum⟩, ⟨Location,Attraction,Museum⟩, ⟨Location,Lake,X⟩, or

$\langle X, X, X \rangle$, where an X indicates that no value at that level applies [18].

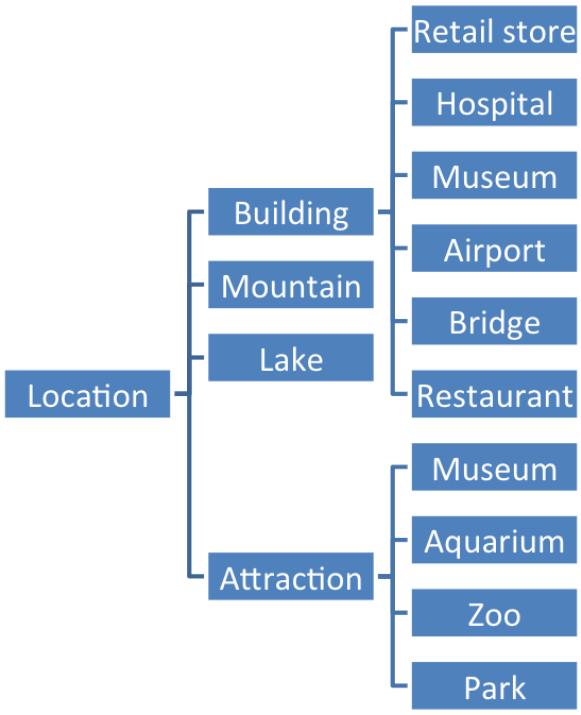


Figure 2.1: A labeling tree based on the text categorization dataset RCV1 [19].

In the active over-labeling setting, each instance U is initially labeled with the vector $\langle ?,?,\ldots,? \rangle$, where ? denotes an unspecified label that is yet to be purchased. A vector of labels is denoted as $\langle \ell_1, \ell_i, \ldots, \ell_k \rangle$. A label ℓ_i is the instance's label at the ith level of the tree. Furthermore, if $j > i$, and $\ell_i = X$ or no value at that level, then $\ell_j = X$ as well, since no other label farther than ℓ_i from the root can be defined. Thus we let ℓ_i denote the largest value for a given instance such that $\ell_i \not\equiv X$. The values $\ell_i, \ldots, \ell_1$ form a path from a leaf to the root of the tree. For a given instance, a value for ℓ_i is purchased at a cost $c_i \geq c_k \geq 0$ for all $i > k$. A purchase of ℓ_i automatically yields the values of ℓ_1 through ℓ_{i-1}. For example a purchase of ℓ_3 for an instance in the RCV1 dataset could yield $\langle$Location,Building,Museum$\rangle$, $\langle$Location,Attraction,Museum$\rangle$, $\langle$Location,Lake,$X\rangle$, or $\langle X, X, X \rangle$. It is assumed that all labels in the same level are distinct, e.g., Museum under Attraction is distinguishable from Museum under Building. A purchase of ℓ_2 for

an instance in the RCV1 dataset could yield ⟨Location,Building,?⟩, ⟨Location,Attraction,?⟩, ⟨Location,Lake,X⟩, or ⟨X, X, X⟩. Note that once an X or a leaf is known for the instance, the rest of the vector labels farther from the root are known to be X. The labeling relationship for an instance for a given class is defined as a function LabelMap(E', m, i) where E' is the label vector, m is the class for which the label is requested, and i is the level in the label hierarchy associated with that class.

2.2 Hierarchical Active Learning

The Hierarchical Active Learning algorithm (HAL) achieves the active over-labeling approach and is shown diagrammatically in *Figure 2.2*. A high level description of the HAL algorithm is given in *Algorithm 1*. Multiple fine-grained classifiers are trained at each level of the hierarchy of the dataset. Every level i, and every class j has a corresponding binary fine-grained classifier $C_{i,j}$. The machine learning algorithm used in $C_{i,j}$ is dependent upon the dataset, this work investigates using SVM and Logit on the protein dataset [1]. The algorithm progresses by purchasing a batch of labels, where the proportion of the total budget b used for a given level i is denoted by a vector p. This step Purchase($b \cdot p_i, i, C_{i,j}(x)$), returns $b \cdot p_i$ worth of label vectors defined up to level i of the labeling tree. The classifier $C_{i,j}$ is used in the purchase function to order the unlabeled instances by the uncertainty, so the instances with the highest uncertainty have their labels purchased in order to maximize the ensemble classifier performance.

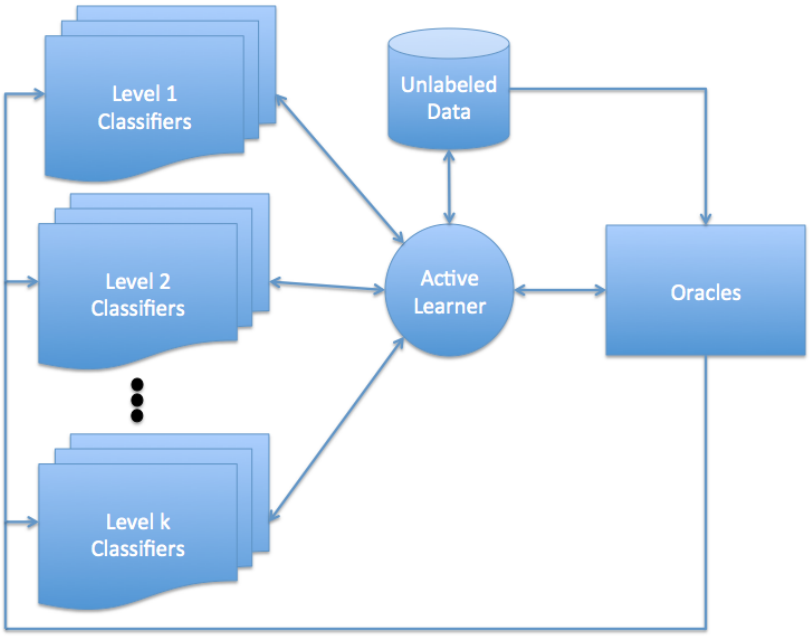


Figure 2.2: Diagram of HAL approach

The classifiers $C_{i,j}$ are combined into an ensemble classifier for the coarse-grained level-1 concept. The level-1 concept $C_{1,*}$ is a disjunction over the concepts j at any given level i, the equation to combine fine-grained classifiers is given in *eqn. 2.1*.

$$C_{i,*}(x) = \max_{s>=i,j} C_{s,j}(x) \tag{2.1}$$

Furthermore the uncertainty at level i is measured with respect to the ensemble classifier $C_{i,*}$. The uncertainty $u_i(x)$ of the label for example x for level i is defined in *eqn. 2.2*.

$$u_i(x) = \frac{1}{2} - \left| C_{i,*} - \frac{1}{2} \right| \quad . \tag{2.2}$$

Algorithm 1 Method hierarchical active learning for a fixed fine-grained ratio (FFR) [18]. See text for Purchase and LabelMap.

function HAL(Unlabeled examples U, labeling tree T, machine learner L, budget B, per-iteration budget b, purchase proportions $p = (p_1, \ldots, p_k)$)
  $E_{i,j} \leftarrow \emptyset$ ▷ binary-labeled train set for level i, label j
  Initialize $C_{i,j}$ for all i, j
  while $B > b$ **do**
    $B \leftarrow B - b$
    for all Level $i \in T$ **do**
      $E' \leftarrow$ PURCHASE$(b\, p_i, i, C_i^*)$
      for all Level $m \leq i$ **do**
        for all Class j in Level m **do**
          $E_{m,j} \cup =$ LABELMAP(E', m, j)
        end for
      end for
    end for
    for all Level $i \in T$ **do**
      for all Class j in Level i **do**
        $C_{i,j} \leftarrow$ Train L on $E_{i,j}$
      end for
    end for
  end while
  return Ensemble classifier
end function

2.3 Dynamically Adapting Purchase Proportions

HAL is a fixed-fine ratio methodology, it takes as input a purchase proportion vector p, which specifies how much of the budget should be used to purchase at a given level in the hierarchy. The following strategy is developed to dynamically adapt to purchase proportions [18]. The task of choosing the level of granularity to purchase labels is framed as a multi-armed bandit problem, and solved using Auer et al.'s greedy bandit algorithm (BANDIT) [20].

For each iteration of purchase, BANDIT chooses a purchasing strategy based on the running average of the observed reward associated with each strategy. The reward or gain for each round is defined in terms of observed model change and the equation is

given in *eqn. 2.3*. Where n is the round number, X is the remaining unlabeled examples $f_j(x_i)$ is HAL's output for the input x_i after the nth round of batch purchases.

$$g(n) = \frac{1}{\|X\|} \sum_{x_i \in X} \log \left(|f_{j-1}(x_i) - f_j(x_i)| \right) \tag{2.3}$$

The gain equation shown in *eqn. 2.3* is modified to further to prevent BANDIT from thrashing between purchasing strategy arms, p and p'. The observed thrashing is resultant from the running average of the $g(n)$ for every strategy slowly decreasing as more instances are obtained. The effect is that the unmodified BANDIT disproportionately favors arms that it has not played recently and have not been recently updated. Thus the following modified BANDIT reward equation is used, it selects between two arms: (1) use the strategy as the previous round, (2) switch strategies. The reward from the (1) is always zero. The reward for (2) is given in *eqn. 2.4*, and is dependent upon the difference in the gain before and after switching. The modified BANDIT reward equation prevents thrashing between arms and solves for the true optimal purchasing strategy.

$$r(n) = \begin{cases} -g(n)/|g(n)| & \text{if } p \to p' \\ g(n)/|g(n)| & \text{if } p' \to p \\ 0 & \text{if } p \to p \text{ or } p' \to p' \end{cases} \tag{2.4}$$

Chapter 3

Related Work

The experiments and methods elicited in this work are the first to demonstrate how leveraging fine-grained label information can improve the accuracy of a coarse-grained (root-level) classifier, and the first investigation of active learning in a hierarchical setting where label acquisition cost can vary [18].

Techniques have been investigated using hierarchies of labels to improve a fine-grained classifier, by backing off to coarse levels of the hierarchy when fine-grained data are sparse. Such techniques have been applied to text classification [21] and rich media indexing [22]. This work presents techniques that work in the opposite direction, utilizing selectively acquired fine-grained labels to improve classification over coarse categories.

Previous work in active learning focused on "pool-based" active learning, where a learner selects instances from a pool of unlabeled data to be labeled by an oracle. Active learning can reduce the expense of purchasing labels by only requesting the most informative labels [23]. Labels that have the highest uncertainty are deemed most informative, uncertainty can be measured in terms of the confidence of output values [24], uncertainty in the parameters of probabilistic models [25], or the size of the model's decision boundary [26]. This work uses uncertainty measured in confidence of output

values and size of the model's decision boundary.

Previous work in active learning has been shown to reduce sampling bias by utilizing the hierarchical structure of input features [27, 28]. This work focuses on active learning over hierarchically structured output labels [18].

3.1 Application to Dispatch Dataset

The analysis of the protein data set presented in this thesis, largely follows Mo et al.'s [18] experiments on the Reuters Corpus Volume I (RCV1) text categorization dataset shown in section 2.1. HAL is applied to a Dispatch dataset by Mo et al. [18]. This dataset contains 375,026 manually labeled hierarchical names across 1,384 newspaper articles [19]. This is a clear example where fine-grained labels have a higher cost since it is relatively easy for a person to manually determine the coarse level question of whether or not the article pertains to an organization. And relatively difficult to determine the fine level question of whether or not the article pertains to a railroad or a zoo. The first analysis step is to confirm that fine-grained classifiers outperform coarse-grained classifiers and active learning outperforms passive learning. These results are shown in *Figure 3.1*.

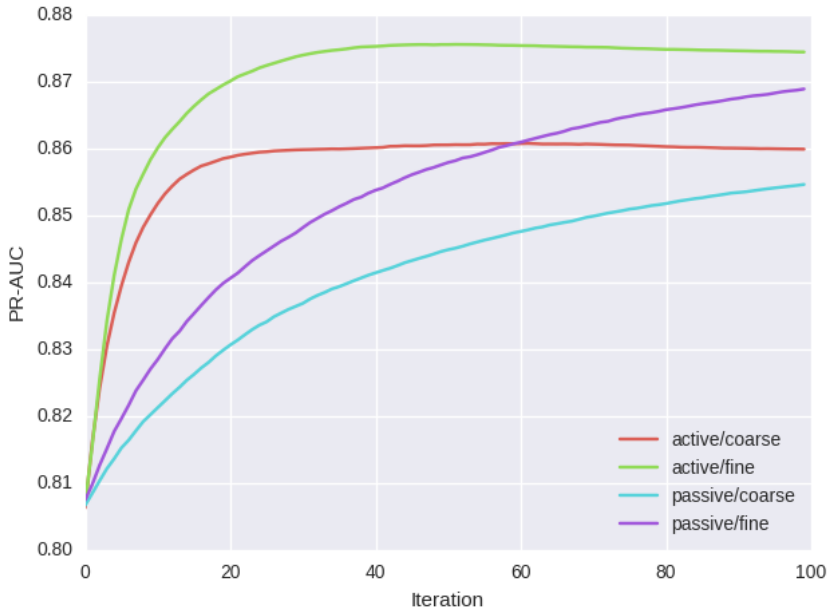
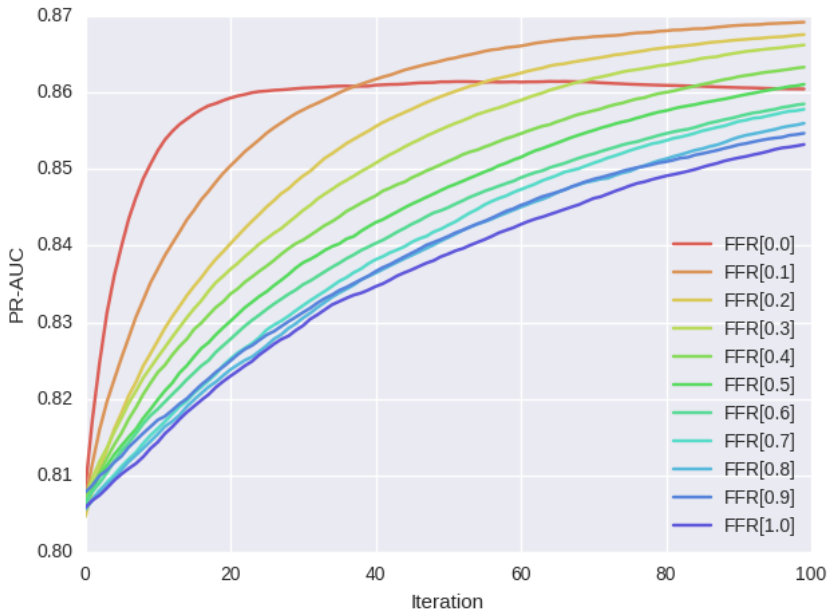
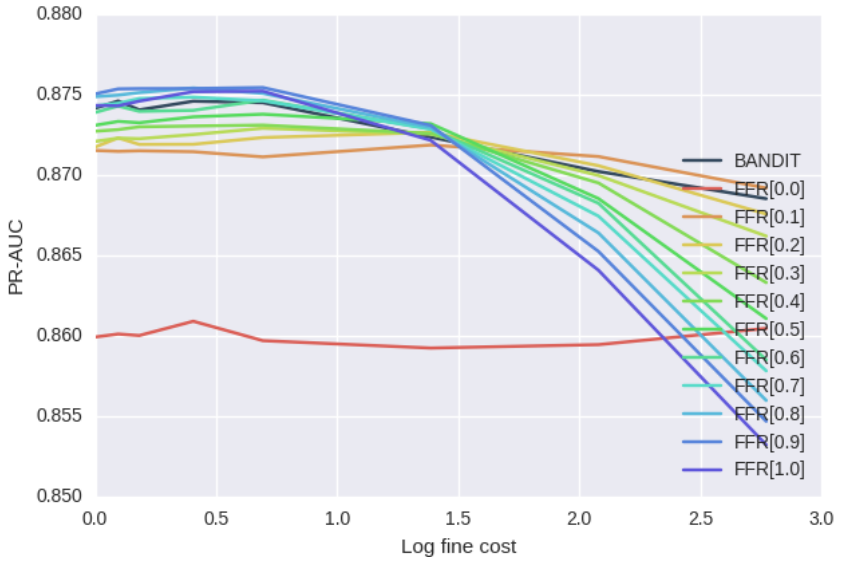


Figure 3.1: Application of HAL demonstrating the benefit of fine-grained active learning on the RCV1 dataset.

The next analysis step is to test HAL with various p proportion ratios. Since the proportion for each fine class remains fixed throughout the rounds of the experiment and have a fixed fine ratio (FFR). The experiments by Mo et al. [18] are shown in *Figure 3.2 (a)*. Furthermore, the Mo et al. performs the BANDIT approach on the RCV1 dataset, these results are shown in *Figure 3.2 (b)*. The results validate the BANDIT approach and confirm that it is robust to differences in label cost and label budget.



(a) HAL FFR experiments on the RCV1 dataset with fine cost of 16 and coarse cost of 1.



(b) BANDIT experiments on the RCV1 dataset, again with fine cost of 16.

Figure 3.2: HAL and BANDIT experiments for RCV1 dataset.

Chapter 4

Experimental Setup

4.1 Training and Testing Coarse-Grain and Fine-Grain Classifiers

The bioinformatics dataset is composed of 9 classes as shown in *Figure 1.1*. The coarse-level concept is whether or not the protein resides within the mitochondria. The negative case of not residing within the mitochondria is class 0. The positive case of residing within the mitochondria corresponds to any of the 8 target compartment classes, numbered 1 through 8. Since the negative case has no fine-grained labels, the fine-grained classifier is composed of separate classifiers for each of the fine-grained labels. The 8 fine-grained classifiers are trained such that only the instances of the class corresponding to that classifier's target compartment are marked as positive, all the others are treated as negative. The coarse-level classifier treats all fine-grained target compartment instances as members of a single positive class. For all classifiers the non mitochondrion instances are treated as negative or 0 labeled. The totals for each class type is shown in *Table 4.1a*. Throughout this experiment a 10 folds cross validation strategy is used, an example

partitioning in shown in *Table 4.1b*.

Table 4.1: This dataset contains 20098 instances total with 449 features each. An example partitioning is shown, some classes like 1 and 5 contain only 1-2 instances in a given test set. Note there is a heavy class imbalance with approx. 20 negative instances for each positive instance.

Classes	Count
0	19136
1	13
2	185
3	324
4	190
5	11
6	104
7	59
8	76
Tot All	20098
Tot Coarse	19136
Tot Fine	962
Features	449

(a) Classes

Folds	All	0	1	2	3	4	5	6	7	8
1	2010	1914	1	19	32	19	1	11	6	7
2	2010	1914	1	19	32	19	1	11	6	7
3	2010	1914	1	19	32	19	1	11	5	8
4	2010	1914	1	19	32	19	1	10	6	8
5	2010	1914	1	18	33	19	1	10	6	8
6	2010	1914	1	18	33	19	1	10	6	8
7	2010	1913	2	18	33	19	1	10	6	8
8	2010	1913	2	18	33	19	1	10	6	8
9	2009	1913	2	18	32	19	2	10	6	7
10	2009	1913	1	19	32	19	1	11	6	7
Total	20098	19136	13	185	324	190	11	104	59	76

(b) Folds

Each partition contains a representative portion of each class, the instances are randomly distributed between partitions. The train set is composed of joining 9 of the partitions together holding 1 fold out for the test set. An example of the totals for a Train and Test set is shown on *Table 4.2*.

Table 4.2: Example of totals for the Train and Test corresponding to when the first fold is held out to be the test set.

Train	All	0	1	2	3	4	5	6	7	8
Total	18088	17222	12	166	292	171	10	93	53	69
Test	All	0	1	2	3	4	5	6	7	8
Total	2010	1914	1	19	32	19	1	11	6	7

Because the experiment will involve running multiple rounds iteratively increasing

the number of instances on which the classifiers are trained and tested, a subset was used to tune the parameters of the classifiers. This allowed variations of the classifier parameters to be run rapidly and for the class weight parameter to be tuned for various round sizes. The reduced subset contains a randomly chosen group of approximately 1/5 of the negatives. The class totals and example partitioning for the reduced subset is shown in *Table 4.3*. After tuning parameters on the subset of data, parameter values are held fixed and experiments are re-run on a new partitioning containing the entire dataset.

Table 4.3: The subset of instances used for tuning classifier parameters contains approximately 1/5 and retains all positive instances.

Classes	Count
0	3827
1	13
2	185
3	324
4	190
5	11
6	104
7	59
8	76
Tot All	4789
Tot Coarse	3827
Tot Fine	962
Features	449

(a) Classes Subset

Folds	All	0	1	2	3	4	5	6	7	8
1	479	383	1	19	32	19	1	11	6	7
2	479	383	1	19	32	19	1	11	6	7
3	479	383	1	19	32	19	1	11	6	7
4	479	383	1	19	32	19	1	11	5	8
5	479	383	1	19	32	19	1	10	6	8
6	479	383	1	18	33	19	1	10	6	8
7	479	383	1	18	33	19	1	10	6	8
8	479	382	2	18	33	19	1	10	6	8
9	479	382	2	18	33	19	1	10	6	8
10	478	382	2	18	32	19	2	10	6	7
Total	4789	3827	13	185	324	190	11	104	59	76

(b) Folds Subset

Table 4.4: Example totals for the train and test set for the subset of data. The subset of data is used for the majority of the parameter search.

Train	All	0	1	2	3	4	5	6	7	8
Total	4310	3444	12	166	292	171	10	93	53	69
Test	All	0	1	2	3	4	5	6	7	8
Total	479	383	1	19	32	19	1	11	6	7

Throughout this project the python library sci-kit learn is used for the implementation of the classification, preprocessing, and evaluation algorithms [5]. The Support Vector Machine (SVM) supervised learning algorithm is used on the un-scaled subset of the data to obtain the base results shown in *Table 4.5*. The coarse and the fine algorithm performance is shown for each of the 10 folds along with the average performance across the 10 folds. Also the receiver operator characteristic and precision recall curves are calculated with fine instances weighted according to the number of of instances in the test set divided by the number of positive instances in the test set which is a value of 4.99 for the data subset.

Table 4.5: SVM default results without parameter selection or preprocessing. Where PR curve AUC is (pr), ROC curve AUC is (roc), accuracy is (acc), F1-measure is (f1).

coarse-pr	fine-pr	coarse-roc	fine-roc	coarse-acc	fine-acc	coarse-f1	fine-f1
0.807	0.796	0.779	0.768	0.816	0.802	0.214	0.021
0.848	0.822	0.828	0.790	0.825	0.804	0.263	0.041
0.846	0.821	0.810	0.765	0.818	0.802	0.243	0.021
0.860	0.832	0.826	0.775	0.831	0.802	0.319	0.021
0.859	0.829	0.828	0.783	0.833	0.804	0.298	0.041
0.796	0.763	0.748	0.715	0.816	0.806	0.214	0.061
0.838	0.825	0.797	0.792	0.818	0.800	0.243	0.020
0.836	0.816	0.803	0.770	0.823	0.800	0.309	0.020
0.863	0.845	0.833	0.805	0.829	0.797	0.305	0.000
0.844	0.806	0.806	0.758	0.836	0.807	0.339	0.061
avg 0.840	avg 0.815	avg 0.806	avg 0.772	avg 0.825	avg 0.802	avg 0.275	avg 0.031

Table 4.6: SVM default results confusion matrix. Where True Negatives is (tn), False Positives is (fp), False Negatives (fn), True Positives is (tp).

coarse-tn	fine-tn	coarse-fp	fine-fp	coarse-fn	fine-fn	coarse-tp	fine-tp
379	383	4	0	84	95	12	1
380	383	3	0	81	94	15	2
378	383	5	0	82	95	14	1
379	383	4	0	77	95	19	1
382	383	1	0	79	94	17	2
379	383	4	0	84	93	12	3
378	382	5	1	82	95	14	1
375	382	7	0	78	96	19	1
379	382	3	0	79	97	18	0
379	382	3	0	75	92	20	3
avg 378.8	avg 382.6	avg 3.9	avg 0.1	avg 80.1	avg 94.6	avg 16.0	avg 1.5

Table 4.7: SVM default condensed view of summary performance metrics, each value is the average of 10 folds.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.840	0.806	0.825	0.275	(378.8 / 80.1)	(3.9 / 16.0)
fine	0.815	0.772	0.802	0.031	(382.6 / 94.6)	(0.1 / 1.5)

The primary metric used to make decisions between alternative parameter choices is the PR-AUC and ROC-AUC. The f-measure and accuracy metrics can be shown to be correlated to a chosen point on the ROC or PR curves. As shown in *Figure 5.2* on *page 47*, each point on the ROC curve has an associated chosen accuracy point, both the coarse and fine classifiers have similar sets of accuracy and f-measure points. The chosen threshold used to output the accuracy, f-measure and confusion matrices varies between the coarse and fine classifier, so at a first glance it appears as if fine out performs coarse in these metrics but an alternative threshold could be selected for the coarse classifier to obtain metrics matching the fine output. Alternatively, the PR-AUC and ROC-AUC compare the correctness of the entire ranking of the instances in the test set by the classifier, and

thus eliminate the need to consider the dynamic tuning of the threshold used by the classifier to output a given confusion matrix, accuracy, and f-measure score. In general as parameter selection in sections 4.1.1-4.1.11 is elicited the choices from previous sections are used in any sections that follow.

4.1.1 Varying SVM Scaling Methods

Different scaling methods are used to preprocess the data [5]. The standard scaling (std-scaler) strategy centers all features around zero with variance in the same order, i.e. it outputs the features with a mean of zero and a unit variance. The minimum maximum scaling (minimax-scaler) strategy scales features between a minimum and maximum value, which is 0 and 1. The normalization scaling (norm-scaler) strategy scales individual samples to have a unit norm. Each preprocessing strategy is applied on the entire dataset before training and testing is performed. Preprocessing is performed with the default kernel option which is Radial Basis Function (RBF). The SVM std-scaler method is discovered to have the highest performance.

Table 4.8: SVM minimax-scaler results.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.881	0.855	0.799	0.000	(382.7 / 96.1)	(0.0 / 0.0)
fine	0.840	0.810	0.799	0.000	(382.7 / 96.1)	(0.0 / 0.0)

Table 4.9: SVM norm-scaler results.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.801	0.791	0.799	0.000	(382.7 / 96.1)	(0.0 / 0.0)
fine	0.636	0.615	0.799	0.000	(382.7 / 96.1)	(0.0 / 0.0)

Table 4.10: SVM std-scaler results. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.912	0.882	0.881	0.631	(372.7 / 47.1)	(10.0 / 49.0)
fine	0.879	0.848	0.809	0.094	(382.7 / 91.3)	(0.0 / 4.8)

4.1.2 Varying SVM Kernels

Different kernel functions were used in the SVM classifier including: Radial Basis Function (RBF), Polynomial Degree 3 and 6 (Poly), Linear, and Sigmoid [5]. The chosen preprocessing strategy of std-scaler is used for these results. The RBF kernel is discovered to have the highest performance.

Table 4.11: Linear kernel results.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.867	0.841	0.853	0.599	(355.5 / 43.4)	(27.2 / 52.7)
fine	0.816	0.789	0.828	0.523	(351.1 / 50.8)	(31.6 / 45.3)

Table 4.12: Poly degree 3 kernel results.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.816	0.817	0.807	0.169	(376.9 / 86.7)	(5.8 / 9.4)
fine	0.755	0.743	0.801	0.063	(380.3 / 92.9)	(2.3 / 3.2)

Table 4.13: Poly degree 6 kernel results.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.659	0.637	0.797	0.037	(379.5 / 94.2)	(3.2 / 1.9)
fine	0.624	0.584	0.794	0.020	(379.0 / 95.1)	(3.7 / 1.0)

Table 4.14: Sigmoid kernel results.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.703	0.693	0.773	0.405	(333.0 / 59.0)	(49.7 / 37.1)
fine	0.653	0.622	0.789	0.127	(370.3 / 88.7)	(12.4 / 7.4)

Table 4.15: RBF kernel results. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.912	0.882	0.881	0.630	(372.6 / 47.1)	(10.1 / 49.0)
fine	0.879	0.848	0.809	0.094	(382.7 / 91.3)	(0.0 / 4.8)

4.1.3 Varying SVM Feature Selection

I tried different feature selection percentages. The Select Percentile library was used from sci-kit learn [5]. This is a univariate feature selection strategy that ranks the features usability for classification according to a statistical measure, then keeps a certain percentage of the features. The 100% of features example is simply the result from the previous section. The 75% feature selection strategy is discovered to have the highest performance. Note that leveraging the fine-grained labels did not improve classifier performance relative to the coarse classifier. An alternative classifier strategy Logistic Regression (Logit) is investigated, starting in the following section 4.1.4.

Table 4.16: SVM select percentile, keep 25% of features.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.907	0.875	0.877	0.623	(370.7 / 47.1)	(12.0 / 49.0)
fine	0.854	0.823	0.806	0.068	(382.7 / 92.7)	(0.0 / 3.4)

Table 4.17: SVM select percentile, keep 50% of features.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.913	0.885	0.879	0.632	(371.3 / 46.4)	(11.4 / 49.7)
fine	0.874	0.842	0.810	0.097	(382.7 / 91.2)	(0.0 / 4.9)

Table 4.18: SVM select percentile, keep 75% of features. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.913	0.883	0.878	0.622	(372.1 / 47.9)	(10.6 / 48.2)
fine	0.880	0.848	0.809	0.089	(382.7 / 91.6)	(0.0 / 4.5)

4.1.4 Varying Logistic Regression Scaling

Testing out the same options for preprocessing scaling that were varied for SVM. The MinMax scaling option is discovered to have the highest performance.

Table 4.19: Logistic Regression - No scaling.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.887	0.862	0.867	0.615	(364.1 / 45.0)	(18.6 / 51.1)
fine	0.854	0.837	0.833	0.395	(372.8 / 69.9)	(9.9 / 26.2)

Table 4.20: Logistic Regression standard scaling.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.864	0.849	0.846	0.583	(353.8 / 44.8)	(28.7 / 51.3)
fine	0.833	0.816	0.831	0.471	(362.0 / 60.2)	(20.5 / 36.0)

Table 4.21: Logistic Regression normalization scaling.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.790	0.761	0.799	0.000	(382.7 / 96.1)	(0.0 / 0.0)
fine	0.767	0.735	0.799	0.000	(382.7 / 96.1)	(0.0 / 0.0)

Table 4.22: Logistic Regression MinMax scaling. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.891	0.867	0.864	0.581	(368.6 / 50.9)	(14.1 / 45.2)
fine	0.888	0.862	0.812	0.130	(382.1 / 89.3)	(0.6 / 6.8)

4.1.5 Varying Logistic Regression Feature Selection

Tested out the same options for feature selection that were varied for SVM. The 100% feature selection strategy is discovered to have the highest performance.

Table 4.23: Logistic Regression select percentile 25%.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.872	0.848	0.849	0.497	(370.8 / 60.3)	(11.9 / 35.8)
fine	0.869	0.845	0.804	0.052	(382.2 / 93.5)	(0.5 / 2.6)

Table 4.24: Logistic Regression select percentile 50%.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.875	0.849	0.849	0.497	(370.8 / 60.3)	(11.9 / 35.8)
fine	0.872	0.846	0.803	0.050	(382.2 / 93.6)	(0.5 / 2.5)

Table 4.25: Logistic Regression select percentile 75%.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.871	0.847	0.848	0.493	(370.6 / 60.6)	(12.1 / 35.5)
fine	0.869	0.845	0.803	0.048	(382.0 / 93.7)	(0.7 / 2.4)

Table 4.26: Logistic Regression select percentile 100%. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.891	0.867	0.864	0.581	(368.6 / 50.9)	(14.1 / 45.2)
fine	0.888	0.862	0.812	0.130	(382.1 / 89.3)	(0.6 / 6.8)

4.1.6 Varying Logistic Regression Positive Class Weight and Cost

Since there is a class imbalance in the dataset, see *Table 4.1a*, class weight and cost parameter pairs are varied. The cost default value is 1.0, and the class weight default value is 1.0. The original value for weighting the fine training instance is the number of instances in the train set divided by the number of positive instances, this is 4.977. The negative instance train weight is always 1.0. The fine weight of 7.5 and Logit cost parameter of 0.1 is discovered to have the most desirable performance, showing an advantage of fine over coarse.

Table 4.27: Logit weight 4.977, cost 1.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.886	0.868	0.787	0.606	(298.7 / 17.9)	(84.0 / 78.2)
fine	0.885	0.862	0.857	0.587	(361.7 / 47.3)	(21.0 / 48.8)

Table 4.28: Logit weight 4.977, cost 0.1

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.880	0.861	0.755	0.579	(280.7 / 15.4)	(102.0 / 80.7)
fine	0.880	0.856	0.851	0.483	(374.2 / 62.7)	(8.5 / 33.4)

Table 4.29: Logit weight 4.977, cost 10.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.876	0.855	0.793	0.603	(304.6 / 21.1)	(78.1 / 75.0)
fine	0.866	0.842	0.835	0.583	(344.8 / 40.9)	(37.9 / 55.2)

Table 4.30: Logit weight 10.0, cost 1.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.883	0.865	0.690	0.536	(245.4 / 10.9)	(137.3 / 85.2)
fine	0.880	0.859	0.822	0.620	(324.2 / 26.7)	(58.5 / 69.4)

Table 4.31: Logit weight 10.0, cost 0.1

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.879	0.863	0.609	0.486	(203.6 / 7.9)	(179.1 / 88.2)
fine	0.881	0.859	0.834	0.621	(334.5 / 31.1)	(48.2 / 65.0)

Table 4.32: Logit weight 10.0, cost 10.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.871	0.851	0.723	0.554	(264.1 / 13.9)	(118.6 / 82.2)
fine	0.861	0.837	0.792	0.585	(309.3 / 26.2)	(73.4 / 69.9)

Table 4.33: Logit weight 7.5, cost 1.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.884	0.867	0.734	0.566	(268.8 / 13.5)	(113.9 / 82.6)
fine	0.882	0.861	0.846	0.624	(343.4 / 34.6)	(39.3 / 61.5)

Table 4.34: Logit weight 7.5, cost 0.1. This option is chosen due to showing advantage for the fine classifier compared to the coarse classifier.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.880	0.862	0.668	0.517	(234.8 / 11.1)	(147.9 / 85.0)
fine	0.881	0.858	0.859	0.613	(357.3 / 42.3)	(25.4 / 53.8)

Table 4.35: Logit weight 7.5, cost 10.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.873	0.852	0.757	0.578	(283.2 / 16.7)	(99.5 / 79.4)
fine	0.863	0.839	0.810	0.588	(323.3 / 31.4)	(59.4 / 64.7)

4.1.7 Varying Logistic Regression Fine Class Weights

The weight for each of the separate fine classes is tuned by multiplying, the class weight of 7.5, determined in the previous section by a fixed ratio. A weight ratio of 1.0 would

output a fine class weight of 7.5. A weight ratio of 0.5 would output a fine class weight of 3.75. Subsections showing the tuning results for each of the 8 fine-grained classes follow. The confusion matrices and output metrics for the individual fine class are shown in order to demonstrate how well the classifier is learning that fine-grained class. These metrics are the average of 10 folds. The coarse classifier output is not shown as it will not vary or be dependent upon the fine class weight tuning.

4.1.7.1 Tune Fine Class 1 Weights

The fine class 1 weight ratio of 3.0 is discovered to have the highest performance.

Table 4.36: Logit Class 1 weight ratio 1.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.881	0.858	0.859	0.613	(357.3 / 42.3)	(25.4 / 53.8)
trainCls-1	0.995	0.999	0.998	0.477	(4297.7 / 7.7)	(0.8 / 4.0)
testCls-1	0.722	0.996	0.997	0.100	(477.4 / 1.2)	(0.1 / 0.1)

Table 4.37: Logit Class 1 weight ratio 0.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.880	0.856	0.859	0.613	(357.4 / 42.3)	(25.3 / 53.8)
trainCls-1	0.994	0.998	0.997	0.142	(4298.5 / 10.8)	(0.0 / 0.9)
testCls-1	0.696	0.995	0.997	0.000	(477.5 / 1.3)	(0.0 / 0.0)

Table 4.38: Logit Class 1 weight ratio 3.0. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.882	0.860	0.859	0.617	(357.1 / 41.7)	(25.6 / 54.4)
trainCls-1	0.995	1.000	0.999	0.854	(4295.8 / 1.0)	(2.7 / 10.7)
testCls-1	0.722	0.997	0.998	0.400	(477.1 / 0.7)	(0.4 / 0.6)

Table 4.39: Logit Class 1 weight ratio 5.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.881	0.859	0.860	0.618	(357.0 / 41.5)	(25.7 / 54.6)
trainCls-1	0.995	1.000	0.999	0.850	(4294.3 / 0.0)	(4.2 / 11.7)
testCls-1	0.722	0.997	0.998	0.513	(476.9 / 0.5)	(0.6 / 0.8)

4.1.7.2 Tune Fine Class 2 Weights

The fine class 2 weight ratio of 1.0 is discovered to have the highest performance.

Table 4.40: Logit Class 2 weight ratio 1.0. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.882	0.860	0.859	0.617	(357.1 / 41.7)	(25.6 / 54.4)
trainCls-2	0.800	0.804	0.952	0.200	(4076.9 / 140.5)	(66.8 / 26.0)
testCls-2	0.655	0.689	0.944	0.081	(450.7 / 17.3)	(9.6 / 1.2)

Table 4.41: Logit Class 2 weight ratio 0.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.882	0.857	0.862	0.618	(359.1 / 42.5)	(23.6 / 53.6)
trainCls-2	0.785	0.787	0.961	0.052	(4139.4 / 161.9)	(4.3 / 4.6)
testCls-2	0.656	0.694	0.960	0.009	(459.4 / 18.4)	(0.9 / 0.1)

Table 4.42: Logit Class 2 weight ratio 1.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.877	0.857	0.855	0.620	(352.5 / 39.3)	(30.2 / 56.8)
trainCls-2	0.806	0.814	0.924	0.263	(3924.1 / 108.1)	(219.6 / 58.4)
testCls-2	0.652	0.684	0.914	0.123	(434.8 / 15.6)	(25.5 / 2.9)

4.1.7.3 Tune Fine Class 3 Weights

The fine class 3 weight ratio of 1.0 is discovered to have the highest performance.

Table 4.43: Logit Class 3 weight ratio 1.0. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.882	0.860	0.859	0.617	(357.1 / 41.7)	(25.6 / 54.4)
trainCls-3	0.846	0.852	0.882	0.401	(3628.6 / 120.7)	(390.0 / 170.9)
testCls-3	0.795	0.803	0.873	0.360	(401.2 / 15.4)	(45.2 / 17.0)

Table 4.44: Logit Class 3 weight ratio 0.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.870	0.852	0.839	0.445	(370.9 / 65.1)	(11.8 / 31.0)
trainCls-3	0.838	0.838	0.929	0.288	(3942.0 / 229.7)	(76.6 / 61.9)
testCls-3	0.792	0.798	0.925	0.246	(437.2 / 26.5)	(9.2 / 5.9)

Table 4.45: Logit Class 3 weight ratio 1.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.879	0.855	0.832	0.626	(331.2 / 28.9)	(51.5 / 67.2)
trainCls-3	0.849	0.859	0.813	0.351	(3288.4 / 74.5)	(730.2 / 217.1)
testCls-3	0.795	0.805	0.804	0.318	(363.3 / 10.6)	(83.1 / 21.8)

4.1.7.4 Tune Fine Class 4 Weights

The fine class 4 weight ratio of 1.5 is discovered to have the highest performance.

Table 4.46: Logit Class 4 weight ratio 1.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.882	0.860	0.859	0.617	(357.1 / 41.7)	(25.6 / 54.4)
trainCls-4	0.937	0.942	0.960	0.531	(4038.6 / 72.9)	(100.6 / 98.1)
testCls-4	0.882	0.902	0.952	0.433	(447.1 / 10.2)	(12.7 / 8.8)

Table 4.47: Logit Class 4 weight ratio 0.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.875	0.852	0.855	0.590	(359.2 / 45.9)	(23.5 / 50.2)
trainCls-4	0.928	0.932	0.965	0.397	(4108.1 / 120.9)	(31.1 / 50.1)
testCls-4	0.878	0.898	0.962	0.320	(456.0 / 14.6)	(3.8 / 4.4)

Table 4.48: Logit Class 4 weight ratio 1.5. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.883	0.861	0.856	0.624	(352.4 / 38.8)	(30.3 / 57.3)
trainCls-4	0.941	0.947	0.936	0.462	(3918.1 / 53.2)	(221.1 / 117.8)
testCls-4	0.886	0.903	0.926	0.382	(432.5 / 8.0)	(27.3 / 11.0)

Table 4.49: Logit Class 4 weight ratio 2.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.880	0.859	0.853	0.627	(348.9 / 36.7)	(33.8 / 59.4)
trainCls-4	0.943	0.950	0.917	0.429	(3817.7 / 36.5)	(321.5 / 134.5)
testCls-4	0.886	0.903	0.906	0.352	(421.8 / 6.8)	(38.0 / 12.2)

4.1.7.5 Tune Fine Class 5 Weights

The fine class 5 weight ratio of 10.0 is discovered to have the highest performance.

Table 4.50: Logit Class 5 weight ratio 1.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.883	0.861	0.856	0.624	(352.4 / 38.8)	(30.3 / 57.3)
trainCls-5	0.940	0.941	0.998	0.000	(4300.2 / 10.0)	(0.0 / 0.0)
testCls-5	0.393	0.681	0.998	0.000	(477.8 / 1.0)	(0.0 / 0.0)

Table 4.51: Logit Class 5 weight ratio 0.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.883	0.861	0.856	0.624	(352.4 / 38.8)	(30.3 / 57.3)
trainCls-5	0.911	0.912	0.998	0.000	(4300.2 / 10.0)	(0.0 / 0.0)
testCls-5	0.389	0.672	0.998	0.000	(477.8 / 1.0)	(0.0 / 0.0)

Table 4.52: Logit Class 5 weight ratio 1.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.883	0.861	0.856	0.624	(352.4 / 38.8)	(30.3 / 57.3)
trainCls-5	0.957	0.958	0.998	0.000	(4300.2 / 10.0)	(0.0 / 0.0)
testCls-5	0.396	0.687	0.998	0.000	(477.8 / 1.0)	(0.0 / 0.0)

Table 4.53: Logit Class 5 weight ratio 5.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.883	0.861	0.856	0.624	(352.4 / 38.8)	(30.3 / 57.3)
trainCls-5	0.990	0.990	0.998	0.374	(4299.8 / 7.6)	(0.4 / 2.4)
testCls-5	0.401	0.694	0.998	0.000	(477.7 / 1.0)	(0.1 / 0.0)

Table 4.54: Logit Class 5 weight ratio 10.0. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.883	0.861	0.855	0.623	(352.1 / 38.8)	(30.6 / 57.3)
trainCls-5	0.996	0.997	0.998	0.609	(4293.4 / 2.7)	(6.8 / 7.3)
testCls-5	0.402	0.696	0.996	0.000	(476.8 / 1.0)	(1.0 / 0.0)

Table 4.55: LogitCls5-Wt20

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.881	0.860	0.854	0.622	(351.6 / 38.7)	(31.1 / 57.4)
trainCls-5	0.998	0.998	0.992	0.355	(4265.8 / 0.5)	(34.4 / 9.5)
testCls-5	0.381	0.616	0.989	0.000	(473.5 / 1.0)	(4.3 / 0.0)

4.1.7.6 Tune Fine Class 6 Weights

The fine class 6 weight ratio of 2.0 is discovered to have the highest performance.

Table 4.56: Logit Class 6 weight ratio 1.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.883	0.861	0.855	0.623	(352.1 / 38.8)	(30.6 / 57.3)
trainCls-6	0.945	0.962	0.976	0.303	(4182.5 / 70.8)	(34.1 / 22.8)
testCls-6	0.892	0.936	0.972	0.191	(463.9 / 8.8)	(4.5 / 1.6)

Table 4.57: Logit Class 6 weight ratio 0.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.882	0.860	0.855	0.622	(352.1 / 38.9)	(30.6 / 57.2)
trainCls-6	0.938	0.956	0.978	0.006	(4216.5 / 93.3)	(0.1 / 0.3)
testCls-6	0.881	0.928	0.978	0.000	(468.3 / 10.4)	(0.1 / 0.0)

Table 4.58: Logit Class 6 weight ratio 2.0. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.884	0.861	0.855	0.627	(350.8 / 37.6)	(31.9 / 58.5)
trainCls-6	0.950	0.967	0.949	0.380	(4023.8 / 26.4)	(192.8 / 67.2)
testCls-6	0.897	0.939	0.945	0.292	(447.0 / 5.0)	(21.4 / 5.4)

Table 4.59: Logit Class 6 weight ratio 3.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.884	0.860	0.850	0.629	(346.6 / 35.5)	(36.1 / 60.6)
trainCls-6	0.952	0.969	0.921	0.335	(3885.8 / 8.3)	(330.8 / 85.3)
testCls-6	0.898	0.940	0.915	0.281	(430.5 / 2.6)	(37.9 / 7.8)

4.1.7.7 Tune Fine Class 7 Weights

The fine class 7 weight ratio of 3.0 is discovered to have the highest performance.

Table 4.60: Logit Class 7 weight ratio 1.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.884	0.862	0.855	0.628	(350.8 / 37.5)	(31.9 / 58.6)
trainCls-7	0.892	0.893	0.988	0.000	(4257.1 / 53.1)	(0.0 / 0.0)
testCls-7	0.648	0.720	0.988	0.000	(472.9 / 5.9)	(0.0 / 0.0)

Table 4.61: Logit Class 7 weight ratio 0.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.884	0.861	0.855	0.627	(350.8 / 37.6)	(31.9 / 58.5)
trainCls-7	0.859	0.857	0.988	0.000	(4257.1 / 53.1)	(0.0 / 0.0)
testCls-7	0.636	0.708	0.988	0.000	(472.9 / 5.9)	(0.0 / 0.0)

Table 4.62: Logit Class 7 weight ratio 3.0. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.885	0.863	0.855	0.632	(350.1 / 36.7)	(32.6 / 59.4)
trainCls-7	0.930	0.939	0.986	0.344	(4234.1 / 37.3)	(23.0 / 15.8)
testCls-7	0.667	0.739	0.983	0.105	(470.1 / 5.4)	(2.8 / 0.5)

Table 4.63: Logit Class 7 weight ratio 5.0

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.883	0.860	0.847	0.628	(344.0 / 34.4)	(38.7 / 61.7)
trainCls-7	0.941	0.953	0.956	0.265	(4086.0 / 18.9)	(171.1 / 34.2)
testCls-7	0.674	0.744	0.948	0.099	(452.3 / 4.5)	(20.6 / 1.4)

4.1.7.8 Tune Fine Class 8 Weights

The fine class 8 weight ratio of 1.0 is discovered to have the highest performance.

Table 4.64: Logit Class 8 weight ratio 1.0. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.886	0.864	0.855	0.632	(350.1 / 36.6)	(32.6 / 59.5)
trainCls-8	0.967	0.978	0.982	0.453	(4199.8 / 36.1)	(42.0 / 32.3)
testCls-8	0.896	0.952	0.978	0.308	(465.7 / 5.2)	(5.5 / 2.4)

Table 4.65: Logit Class 8 weight ratio 0.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.885	0.862	0.855	0.630	(350.4 / 37.0)	(32.3 / 59.1)
trainCls-8	0.961	0.972	0.984	0.253	(4229.6 / 56.7)	(12.2 / 11.7)
testCls-8	0.893	0.952	0.982	0.135	(469.5 / 6.8)	(1.7 / 0.8)

Table 4.66: Logit Class 8 weight ratio 1.5

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
fine	0.886	0.864	0.855	0.632	(349.5 / 36.4)	(33.2 / 59.7)
trainCls-8	0.967	0.980	0.978	0.478	(4169.7 / 24.3)	(72.1 / 44.1)
testCls-8	0.892	0.947	0.973	0.376	(462.2 / 3.9)	(9.0 / 3.7)

4.1.8 Varying Logistic Regression Tolerance

There is an additional Logistic parameter for determining a tolerance for the stopping criteria. The default tolerance is 0.0001. The tolerance setting of 0.00001 is discovered to have the highest performance.

Table 4.67: Logit results after fine tuning, effectively had a tolerance of 0.0001

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.880	0.863	0.668	0.517	(234.8 / 11.1)	(147.9 / 85.0)
fine	0.886	0.864	0.855	0.632	(350.1 / 36.6)	(32.6 / 59.5)

Table 4.68: Logit Tolerance 0.0001, notice that the fine pr and roc decreased by 0.001, and that the coarse roc decreased by 0.001 upon rerunning, there is some statistical variation in these metrics.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.880	0.862	0.668	0.518	(234.9 / 11.1)	(147.8 / 85.0)
fine	0.885	0.863	0.855	0.632	(350.1 / 36.7)	(32.6 / 59.4)

Table 4.69: Logit Tolerance 0.00001. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.880	0.863	0.668	0.517	(234.7 / 11.1)	(148.0 / 85.0)
fine	0.886	0.864	0.855	0.632	(350.1 / 36.6)	(32.6 / 59.5)

Table 4.70: Logit Tolerance 0.000001

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.880	0.862	0.668	0.517	(234.8 / 11.1)	(147.9 / 85.0)
fine	0.885	0.863	0.855	0.632	(350.1 / 36.7)	(32.6 / 59.4)

4.1.9 Varying Sample Weight On Test Set and Dropping Intermediate ROC Curve Values

The sample weight, as stated previously, weights fine instances in the ROC and PR curves by the ratio of total number of instances in the test set divided by the total number of positives in the test set. This weighting is performed identically on the coarse and fine classifier. The ROC curve library has a parameter to determine whether or not to drop some suboptimal thresholds which do not appear on a plotted ROC curve [5]. The default setting is to drop intermediate values True, which has the counterintuitive result of a roc curve having on the order of 150 points even though 497 points are passed to the roc curve library method. If drop intermediate values is set to false then the full 497 points

are returned in the calculated roc curve. The default options of using sample weights and dropping intermediate values are discovered to have the highest performance.

Table 4.71: Logit sample weights, drop intermediate values True. The default option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.880	0.862	0.668	0.517	(234.8 / 11.1)	(147.9 / 85.0)
fine	0.885	0.863	0.855	0.632	(350.1 / 36.7)	(32.6 / 59.4)

Table 4.72: Logit no sample weights, drop intermediate values True

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.649	0.862	0.668	0.517	(234.8 / 11.1)	(147.9 / 85.0)
fine	0.663	0.863	0.855	0.632	(350.1 / 36.7)	(32.6 / 59.4)

Table 4.73: Logit sample weights, drop intermediate values False

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.880	0.862	0.668	0.517	(234.8 / 11.1)	(147.9 / 85.0)
fine	0.885	0.863	0.855	0.632	(350.1 / 36.7)	(32.6 / 59.4)

Table 4.74: Logit no sample weights, drop intermediate values False

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.649	0.862	0.668	0.517	(234.8 / 11.1)	(147.9 / 85.0)
fine	0.663	0.863	0.855	0.632	(350.1 / 36.7)	(32.6 / 59.4)

4.1.10 Varying Logistic Regression Positive Class Weight For Full Dataset

The fine class weight for the subset of data is determined be to 7.5, this value should change and be linearly dependent upon the number of instances in the training set. The

weight for the fine class is tuned using all of the data, the original value is the total number of instances in the train set divided by the total number of positives in the train set, which evaluates to 20.887. The previously determined fine class ratios are used in this analysis. The value selected is 23, this value along with 7.5 and the original values of 20.887 and 4.977 for a line with two points that define a function to map a weight original input to a new tuned weight output for all training set sizes. The fine weight of 23 is discovered to have the highest performance when the entire dataset is used.

Table 4.75: Logit entire dataset, weight 20.887

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.867	0.868	0.803	0.280	(1537.6 / 19.2)	(376.0 / 76.9)
fine	0.871	0.868	0.919	0.404	(1792.3 / 41.0)	(121.2 / 55.1)

Table 4.76: Logit entire dataset, weight 23.0. This option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.870	0.871	0.787	0.268	(1503.2 / 17.8)	(410.4 / 78.3)
fine	0.875	0.871	0.913	0.403	(1776.5 / 37.3)	(137.1 / 58.8)

Table 4.77: Logit entire dataset, weight 25.0.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.867	0.868	0.772	0.256	(1473.0 / 17.3)	(440.6 / 78.8)
fine	0.871	0.868	0.905	0.389	(1758.8 / 35.6)	(154.8 / 60.6)

4.1.11 Varying SVM Cost and Gamma

After the Logit classifier is tuned with class weights, the SVM is ran again with the class weights determined by the Logit classifier and a slight advantage for the fine-grained classifier is demonstrated with the SVM as well. The SVM parameters for the rbf kernel

of cost and gamma are varied. The cost is related to a penalty parameter for the error term and gamma is the kernel coefficient and determines the relative significance a single instance can have. The default gamma setting is 0.002967 or (1/num-features) or (1/337). Default cost is actually 1.0, and the default class weight is balanced which weights each class by the number of instances it has in the train set, the same fine class weights used in the LogReg classifier are used in the SVM classifier instead of the SVM's default balanced option. The SVM cost of 0.15 and gamma of 0.0029674 is discovered to have the highest performance.

Table 4.78: SVM Cost 1.0 Gamma 0.0029674

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.901	0.874	0.846	0.651	(336.0 / 27.2)	(46.7 / 68.9)
fine	0.896	0.865	0.871	0.598	(371.1 / 50.1)	(11.6 / 46.0)

Table 4.79: SVM Cost 2.0 Gamma 0.0029674

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.903	0.873	0.866	0.672	(348.9 / 30.4)	(33.8 / 65.7)
fine	0.890	0.857	0.865	0.554	(373.8 / 55.8)	(8.9 / 40.3)

Table 4.80: SVM Cost 0.1 Gamma 0.0029674

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.892	0.869	0.664	0.518	(231.5 / 9.8)	(151.2 / 86.3)
fine	0.899	0.870	0.868	0.623	(363.5 / 43.8)	(19.2 / 52.3)

Table 4.81: SVM Cost 0.05 Gamma 0.0029674

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.883	0.860	0.591	0.474	(194.9 / 8.0)	(187.8 / 88.1)
fine	0.884	0.853	0.858	0.544	(370.1 / 55.5)	(12.6 / 40.6)

Table 4.82: SVM Cost 0.15 Gamma 0.0029674. This cost option is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.896	0.873	0.714	0.553	(257.5 / 11.6)	(125.2 / 84.5)
fine	0.902	0.874	0.871	0.640	(362.1 / 41.1)	(20.6 / 55.0)

Table 4.83: SVM Cost 0.2 Gamma 0.0029674.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.899	0.875	0.755	0.584	(279.6 / 14.0)	(103.1 / 82.1)
fine	0.903	0.875	0.871	0.640	(362.1 / 41.2)	(20.6 / 54.9)

Table 4.84: SVM Cost 0.15 Gamma 0.002. This option for Cost and Gamma is chosen.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.894	0.871	0.706	0.545	(253.6 / 11.7)	(129.1 / 84.4)
fine	0.906	0.877	0.869	0.646	(358.3 / 38.5)	(24.4 / 57.6)

Table 4.85: SVM Cost 0.15 Gamma 0.001

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.883	0.864	0.664	0.516	(232.1 / 10.3)	(150.6 / 85.8)
fine	0.900	0.872	0.868	0.641	(358.5 / 39.2)	(24.2 / 56.9)

Chapter 5

Results and Analysis

5.1 SVM and Logit Classifier Performance

Results are presented running both Logit and SVM classifiers on the entire dataset see *Tables 5.1-5.2*. Both the SVM and the Logit classifiers show a slight advantage for the fine classifier over the coarse classifier in terms of the PR-AUC metric. The ROC-AUC metric is close to identical between fine and coarse for both classifiers, a slight advantage of 0.002 exists for the fine classifier in the SVM classifier.

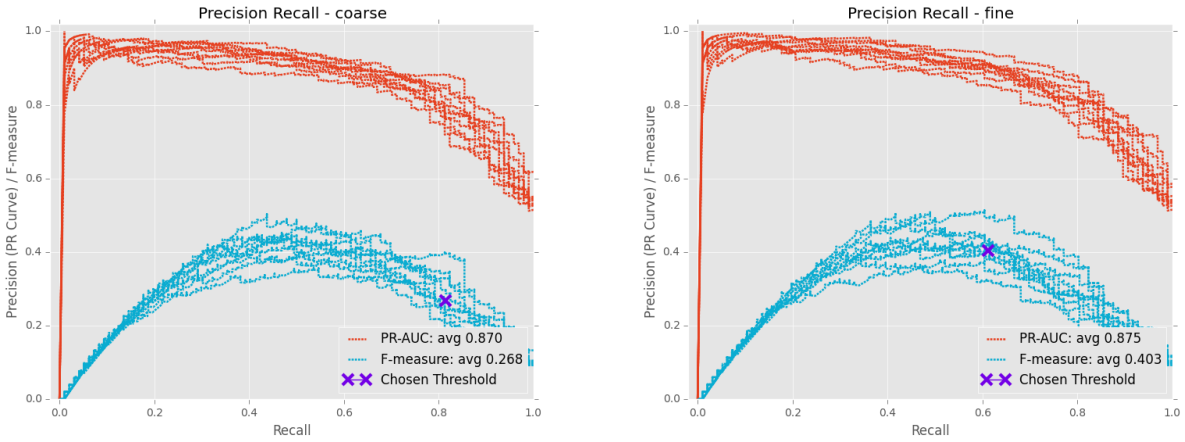
Table 5.1: Logit entire dataset results after parameter tuning

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.870	0.871	0.787	0.268	(1503.2 / 17.8)	(410.4 / 78.3)
fine	0.875	0.871	0.913	0.403	(1776.5 / 37.3)	(137.1 / 58.8)

Table 5.2: SVM entire dataset results after parameter tuning

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.892	0.880	0.866	0.347	(1669.5 / 24.8)	(244.1 / 71.3)
fine	0.898	0.882	0.942	0.485	(1839.0 / 41.5)	(74.6 / 54.6)

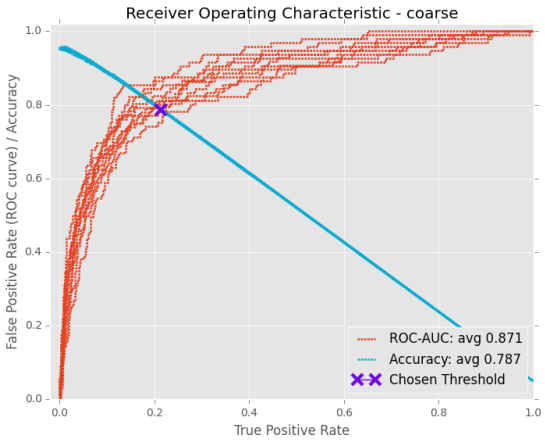
The coarse classifier in both the Logit and SVM classifier has a greater amount of false positives at the default threshold. A further examination of these values is shown in *Figures 5.1-5.2* and *Figures 5.3-5.4*. The figures plot the PR and the ROC curves for each of the 10 folds. Each point on the PR and ROC curve has a corresponding F-measure or accuracy value, these values are plotted on the graphs as a blue line. The graphs demonstrate that the coarse and fine classifiers have close to equivalent average AUC, on the order of 0.007 max difference between fine and coarse. At the default threshold the fine appears to outperform coarse for accuracy and F-measure metrics, but inspection of the plots shows that a coarse threshold can be chosen to match the fine output for both accuracy and F-measure. The PR-AUC does show a slight advantage for fine, which warrants application of the HAL algorithm and Active over labeling approach on this dataset.



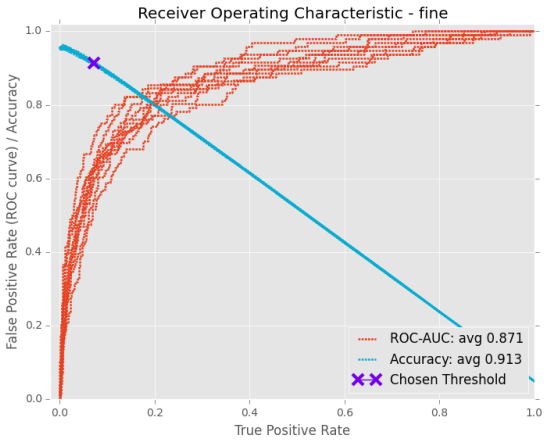
(a) Log Reg Pr Curves - Coarse

(b) Log Reg Pr Curves - Fine

Figure 5.1: The fine default threshold occurs at a point on the PR curve associated with a higher F-measure score compared to the coarse curves.

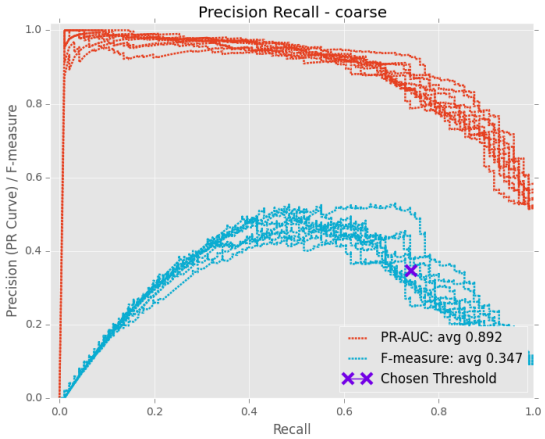


(a) Log Reg ROC Curves - coarse

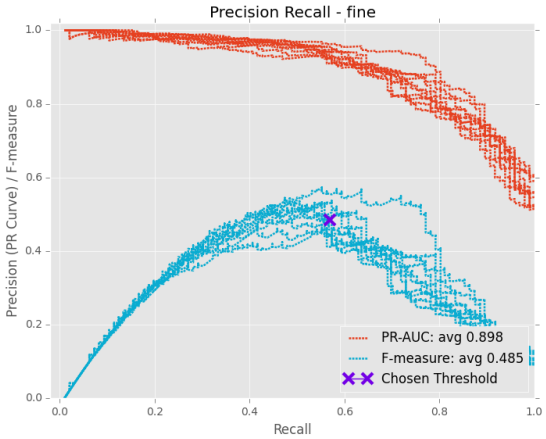


(b) Log Reg ROC Curves - fine

Figure 5.2: Fine has a higher accuracy than coarse at the default threshold for the Logit classifier.

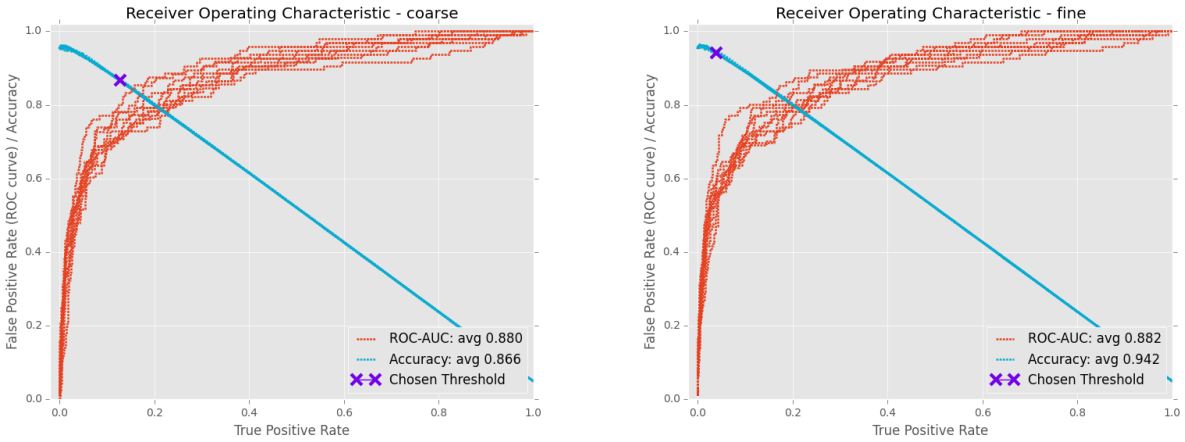


(a) SVM Pr Curves - Coarse



(b) SVM Pr Curves - Fine

Figure 5.3: SVM results for PR curves and F-measure have coarse and fine picking different parts of the curves for their respective thresholds. This results in a slight advantage for fine at the default threshold, similar to the results for the Logit classifier.



(a) SVM ROC Curves - Coarse

(b) SVM ROC Curves - Fine

Figure 5.4: SVM accuracy results are similar between coarse and fine.

5.2 Active vs Passive curves

The plots in *Figures 5.5-5.9* were obtained with a round batch size of 100 and a starter set of 1040 instances out of the total 20098 instances. The plots are the average of 10 folds, for each fold a test set of 2010 instances is used. The test set remains constant throughout the rounds and contains a representative proportion of each of the classes. The starter set is chosen out of the remaining 18088 and it also contains representatives from each class in proportion to that class's prominence in the dataset. The 17048 non-test set, non-starter set instances are added to the training set in batches of 100. This results in total of 171 rounds, 170 batch selecting rounds and 1 starter set round. The Passive approach selects 100 random instances and adds them to the train set. The Active approach runs the classifier on the eligible instances, orders them by their uncertainty and adds the 100 most uncertain instances to the train set. Coarse and fine classifiers share the same starter set. During each round, coarse and fine classifiers are trained on their corresponding sets, which are independent of one another, metrics are outputted on the held out test set which is the same for both coarse and fine.

5.2.1 Plots for Logistic Regression Active vs Passive curves

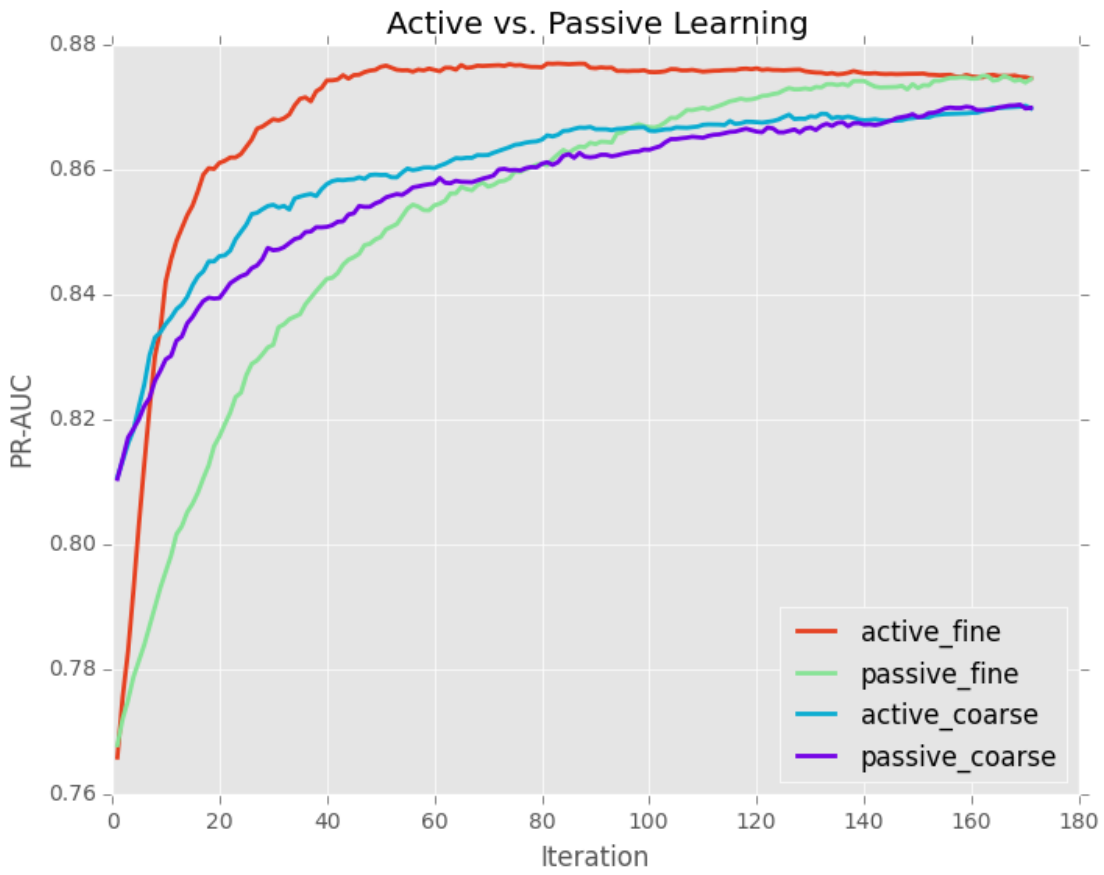


Figure 5.5: The PR-AUC curves for rounds with the Logistic Regression classifier conforms to expectations, with active fine having the highest performance, and Active outperforming Passive for both coarse and fine classifier types.

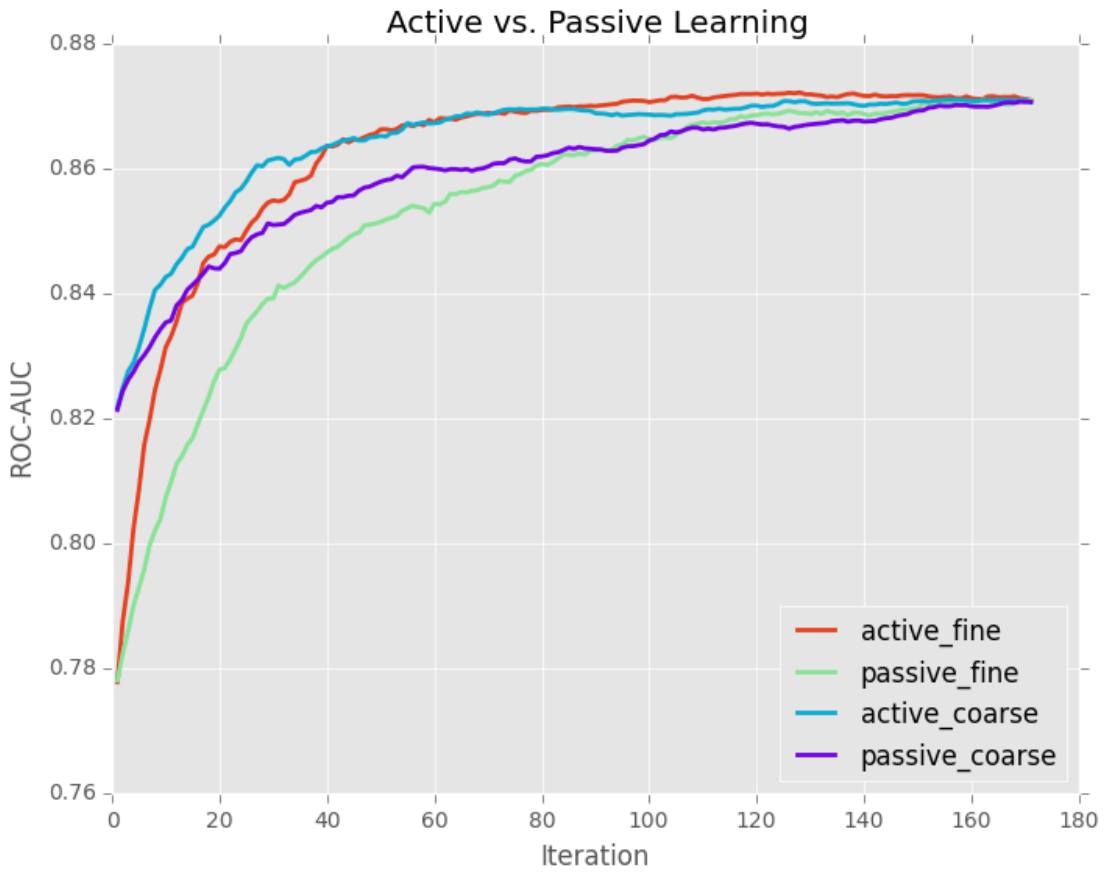
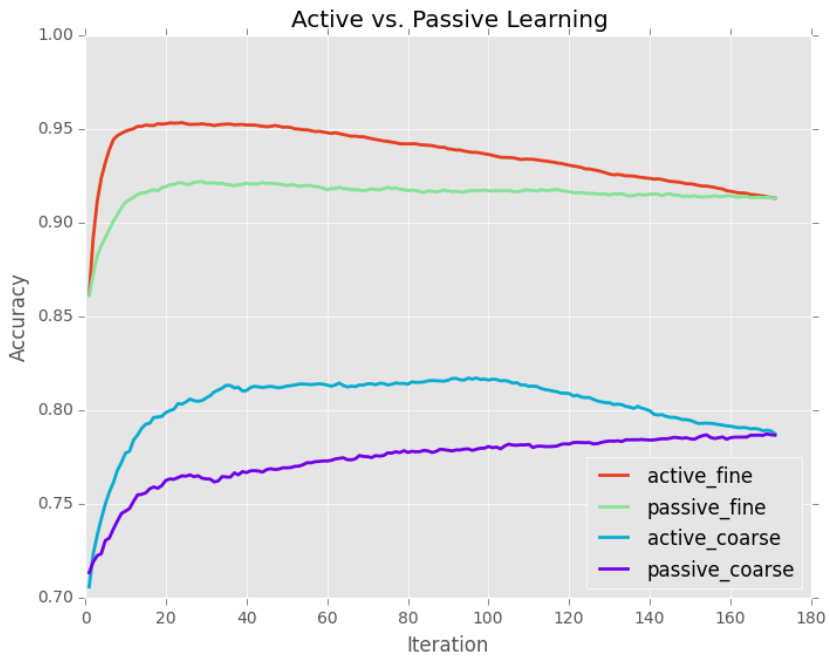
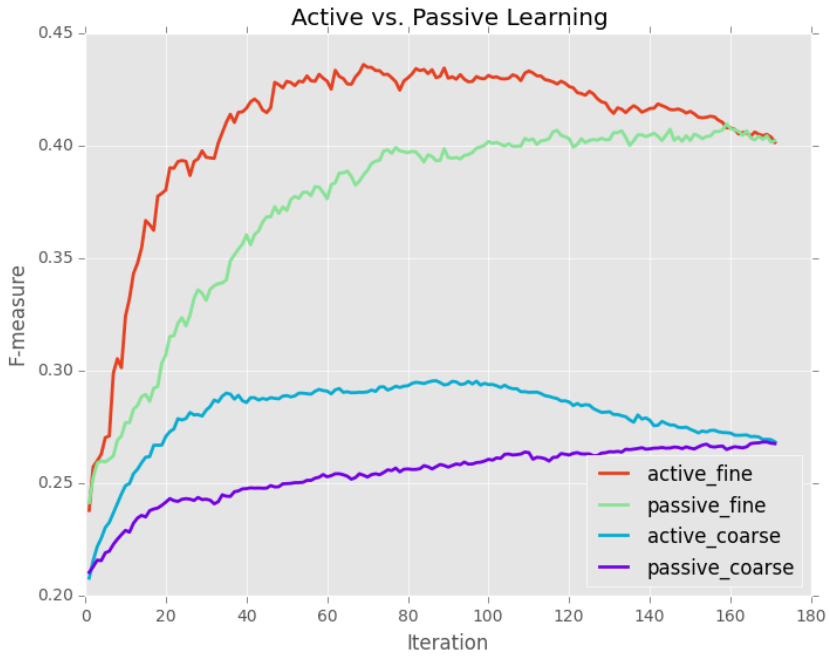


Figure 5.6: The ROC-AUC curves for rounds with the Logistic Regression classifier. The active curves beat out the passive curves for both coarse and fine. Note that active fine ROC curve doesn't converge to the active coarse ROC curve until round 40. This is contrasted to a dominance of the active fine PR curve after round 10.



(a) Logit accuracy



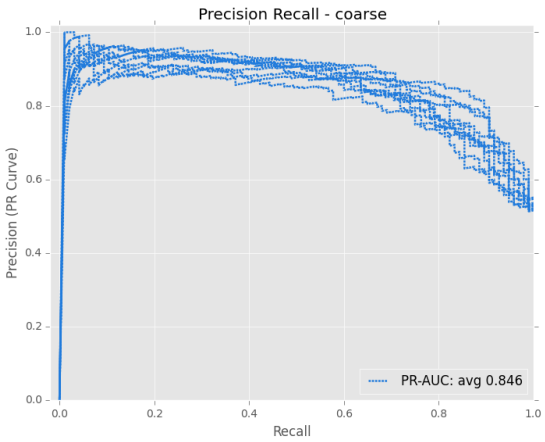
(b) Logit F-measure

Figure 5.7: The accuracy of the classifiers stays at roughly the same rate throughout the rounds; this is due to an effective weighting scheme. Both curves show a dominance of fine over coarse and Active over Passive.

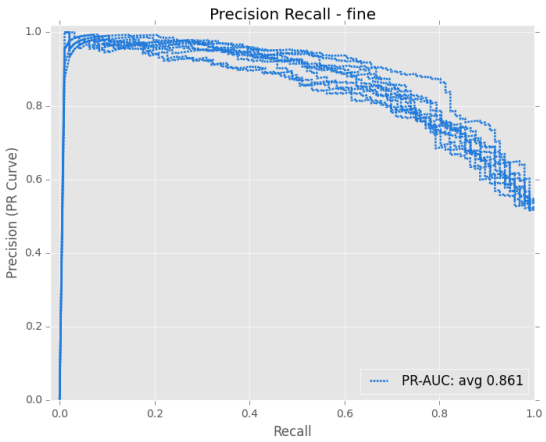
Note that the Active Fine PR-AUC curve surpasses active coarse after round 10 while the active fine ROC-AUC curve is still well below the active coarse at that round. These curves are shown in *Figures 5.8- 5.9*. This is counter-intuitive, because according to a proof in Davis [29] "For a fixed number of positive and negative examples, one curve dominates a second curve in ROC space if and only if the first dominates the second in Precision-Recall space". The theorem uses the following definition of dominance: that every value in the first curve is above the corollary value in second curve. The correlation between PR and ROC curves is that Recall in the PR curve is equivalent to the True Positive Rate in the ROC curve. The average PR-AUC concept is different than that of a plot of PR curves for a round, but if all of the PR curves for fine dominate the curves for coarse then we would expect all of the ROC curves for fine to dominate the ROC curves for coarse and both the ROC-AUC and PR-AUC averages for fine to be greater than that for coarse. However it is shown in *Figure 5.8* that the PR curves for fine do not completely dominate the PR curves for coarse, and similarly for the ROC curves in *Figure 5.9*. Active fine PR-AUC curve does not satisfy the theorem's definition of dominance, since each individual ROC and PR curve contains intersection points between coarse and fine. Thus given that the average PR-AUC for fine is great at round 20 than average coarse PR-AUC, this relationship is not expected to hold between the average ROC-AUC curves.

According to Davis [29], a large change in the number of false positives can still correlate to only only a small change in the number of true positives and thus not affect ROC curve performance. However, Davis states, "Precision, on the other hand, by comparing false positives to true positives rather than true negatives, captures the effect of the large number of (incorrectly classified) negative examples on the algorithm's performance" [29]. Since our dataset demonstrates a heavy class imbalance with a roughly 1:20 ratio of positive to negative instances, the algorithm's ability to classify negative instances should be taken into account when considering overall classifier

performance. The PR curve's ability to capture the effect of an increased number of false positives, reveals the advantage that the fine classifier has over the coarse classifier. This justifies purchasing fine-grained labels over coarse-grained labels to improve classifier performance.

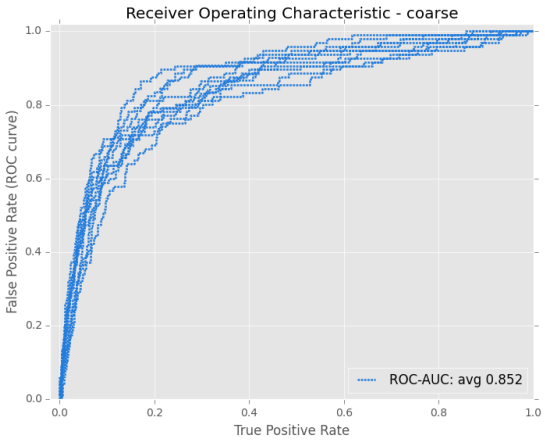


(a) Coarse PR curves at Round 20

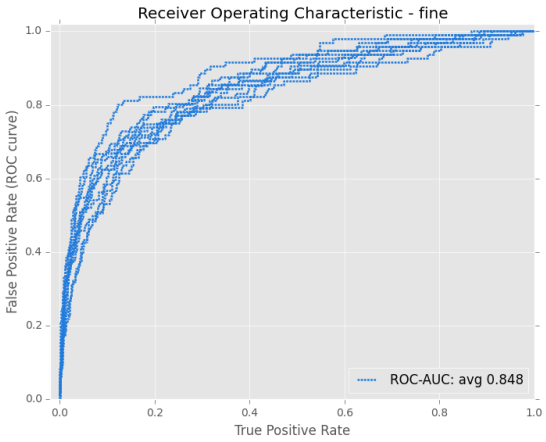


(b) Fine PR curves at Round 20

Figure 5.8: PR curves for each fold at Round 20



(a) Coarse ROC curves at Round 20



(b) Fine ROC curves at Round 20

Figure 5.9: ROC curves for each fold at Round 20

5.2.2 Plots for SVM Active vs Passive curves

The SVM Active vs Passive experiment is performed with the same methodology as the previous section detailed with the exception that a SVM classifier is substituted for the Logit classifier. Due to the greater advantage of average PR-AUC in the Logit classifier, the SVM is not used in the Fixed fine ratio experiments in section 5.3 Plots for Fine Fixed Ratio (FFR) experiments.

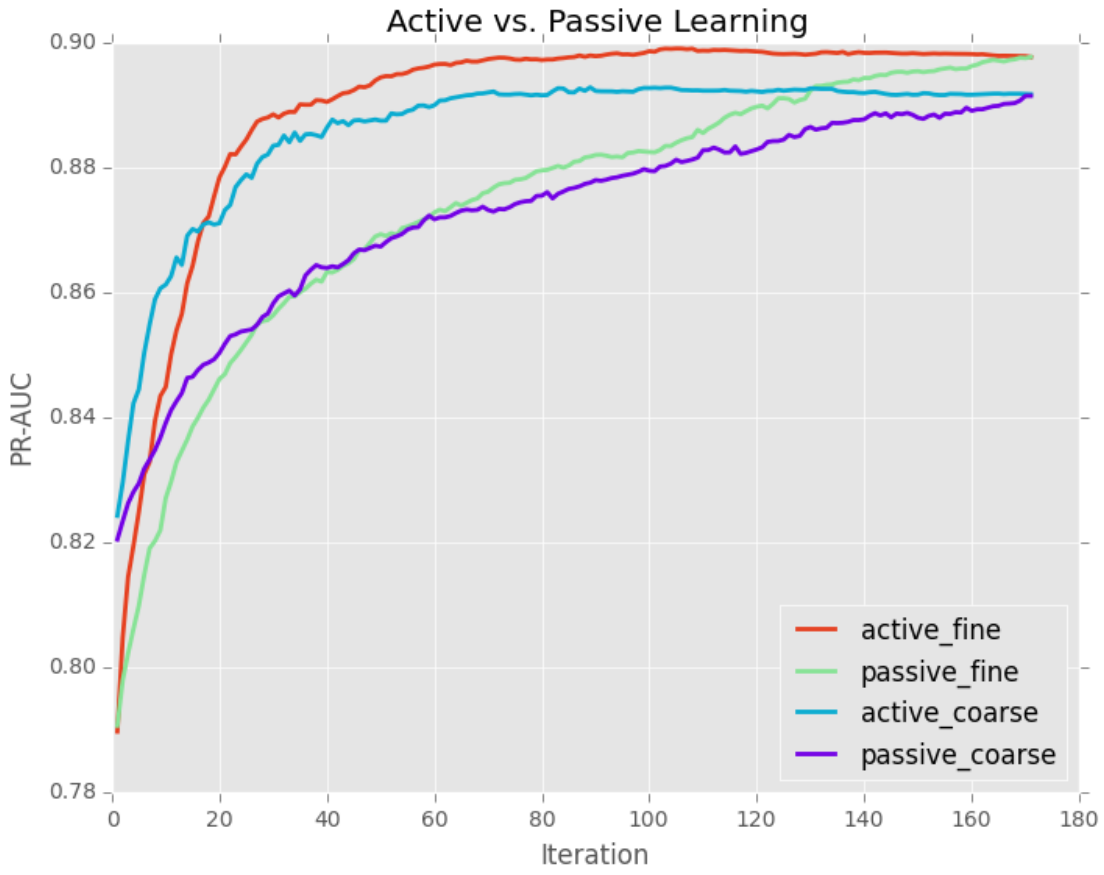


Figure 5.10: The PR AUC curves for SVM show a slight advantage for active fine, similar to the Logit results.

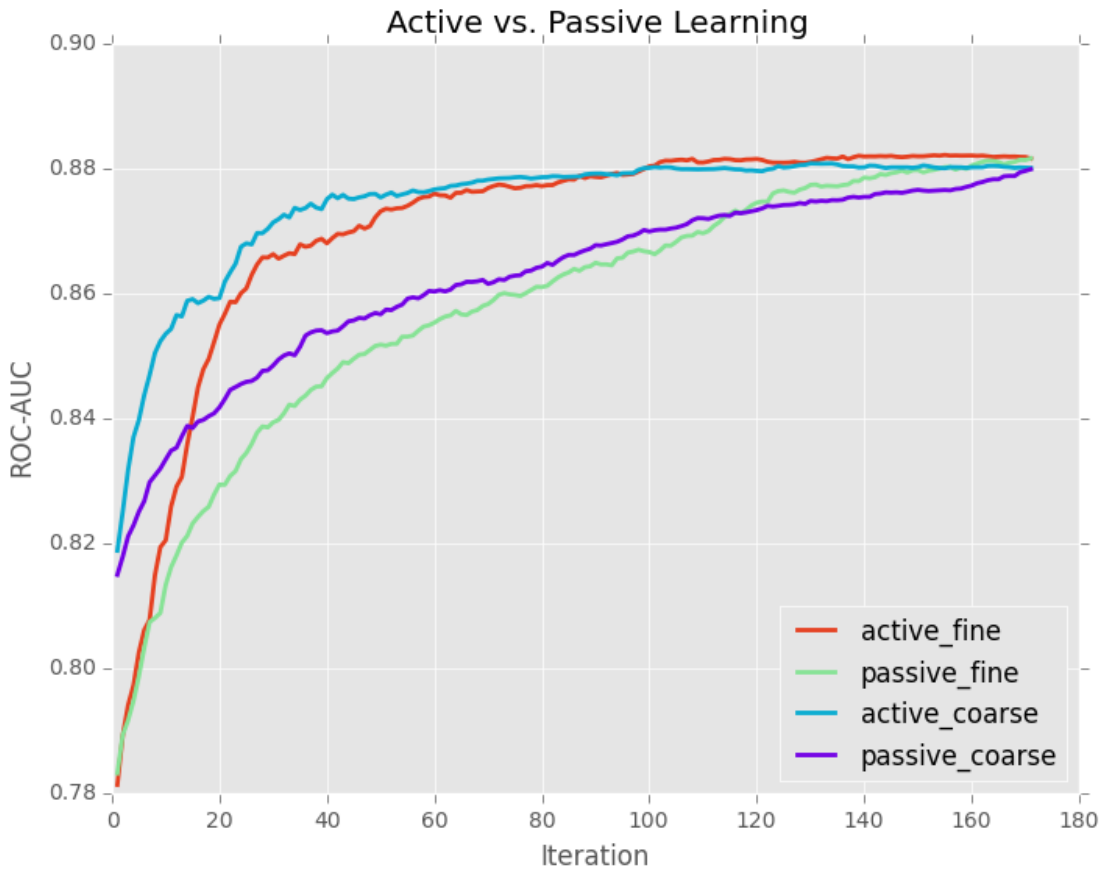


Figure 5.11: The ROC AUC curves for SVM match the Logit results, the convergence of active fine to active coarse takes slightly longer, round 60 compared to round 40.

5.3 Plots for Fine Fixed Ratio experiments

The strategy is changed from purchasing a set number of instances per round to having a set budget per round and spending a portion of that budget on fine and coarse grained labels. The Fine Fixed Ratio (FFR), ranges from 0.0 to 1.0 in increments of 0.1. Note that the FFR 0.0 should roughly correlate to the active coarse curve shown in *Figure 5.5*. Likewise the active fine curve should roughly correlate to the FFR 1.0 curve. However, the correlation is not exact since the FFR experiments use a combination classifier, it trains fine and coarse classifiers on a starter set of the same size and proportion as used in the Logit Active vs Passive experiment, then uses the confidence of both of those classifiers and the end prediction is the max of the two classifiers. Thus even for the FFR 0.0 and FFR 1.0 the starter set trained fine or coarse classifier still contributes to the PR-AUC curve even at the final round 180. The results are an average of 10 folds.

To determine the number of instances to purchase each round, the FFR proportion vector p is multiplied by the round budget of 100. The coarse labels are purchased at a cost of 1.0. The cost of the fine labels will vary, experiments are performed for fine costs of 1,2,4,8,16. After discussions with Cui et al.'s group [1], a reasonable estimate for the fine label cost for the protein data set is around 10, due to the increased number of features needed to distinguish a protein's target compartment. An example of this cost breakdown is as follows, if the fine cost is 8 and p is 0.5, then 50 labels are purchased for coarse and 6.25 labels are purchased for fine. The 0.25 of a fine instance is resolved by purchasing an extra fine label with the probability of 25%. There is a 25% chance for any round to purchase an extra fine label. The round size for the FFR 1.0 curve is relatively small, with at most 13 labels purchased per iteration.

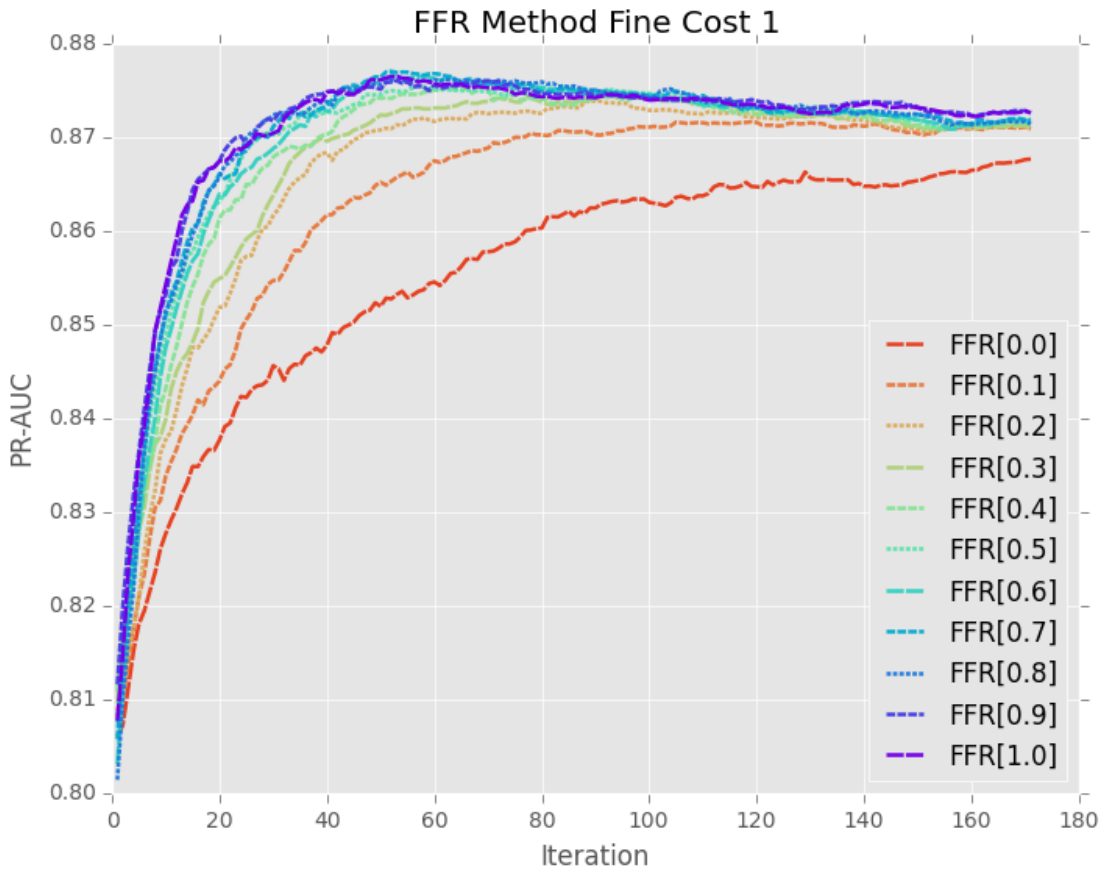


Figure 5.12: For this curve the fine and coarse grain labels both have a cost of 1. The purple 1.0 curve shows that if only fine-grained labels are purchased, the highest performing PR-AUC can be obtained. All FFR ratios end at the same round since the cost of the fine and coarse instances is the same the budget

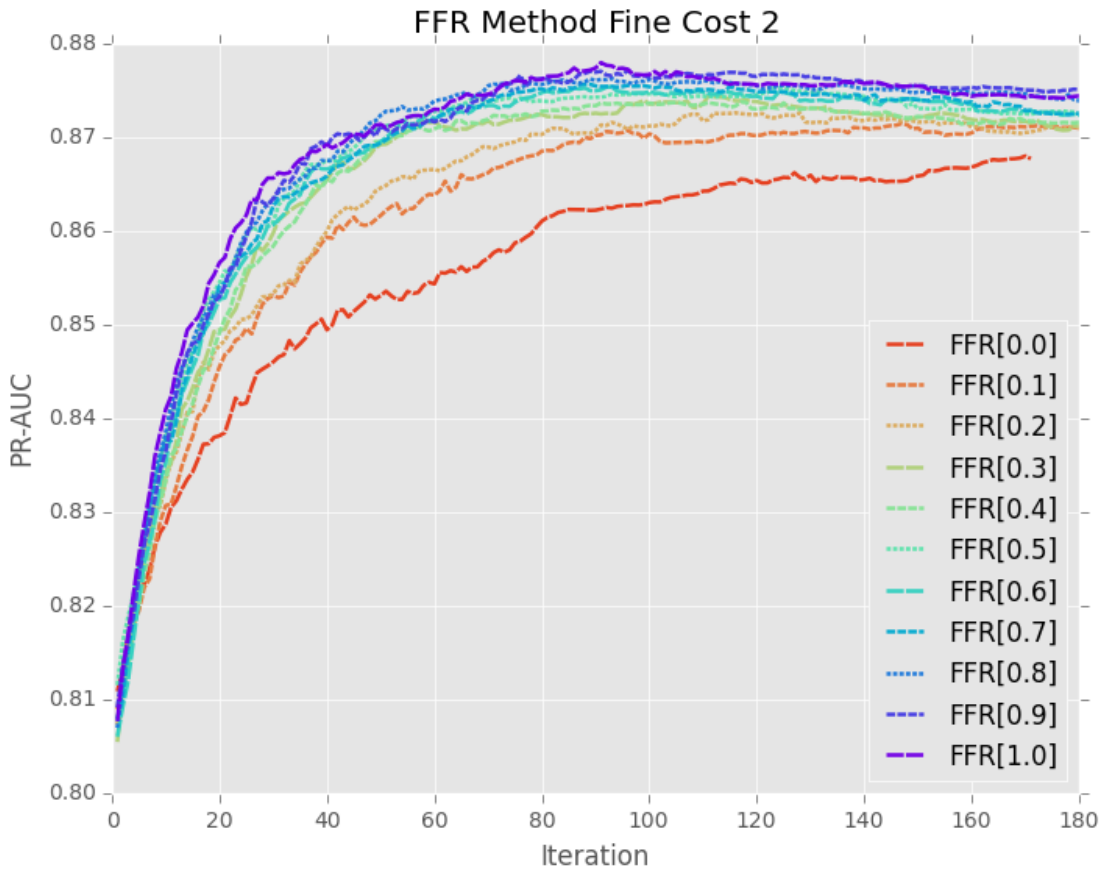


Figure 5.13: At fine cost 2, advantage of the higher FFR values decreases but the ordering of the curves remains unchanged.

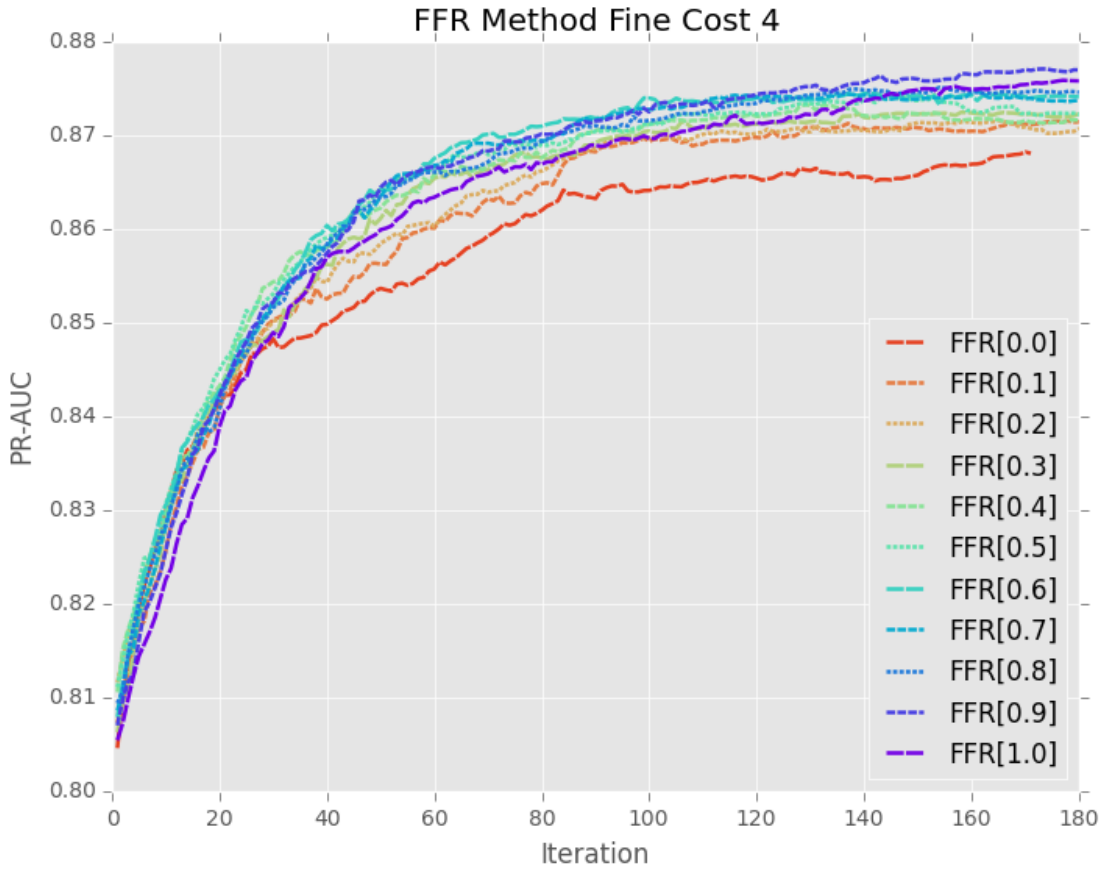


Figure 5.14: At fine cost 4, the highest FFR 1.0 is no longer preferred, the cost is to high for fine instances PR-AUC utility to overcome the PR-AUC increase gained by purchasing more coarse instances.

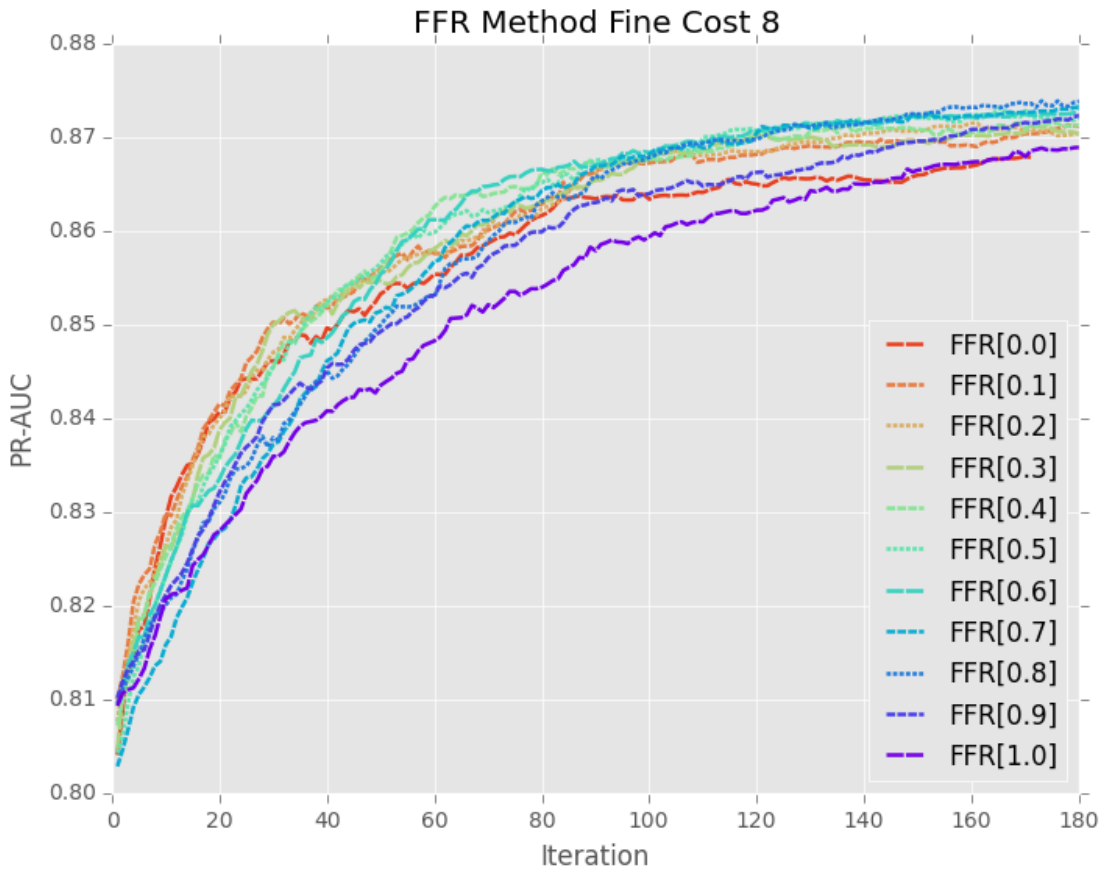


Figure 5.15: At fine cost 8 the middle FFR values outperform the extreme values for rounds 0 to 180.

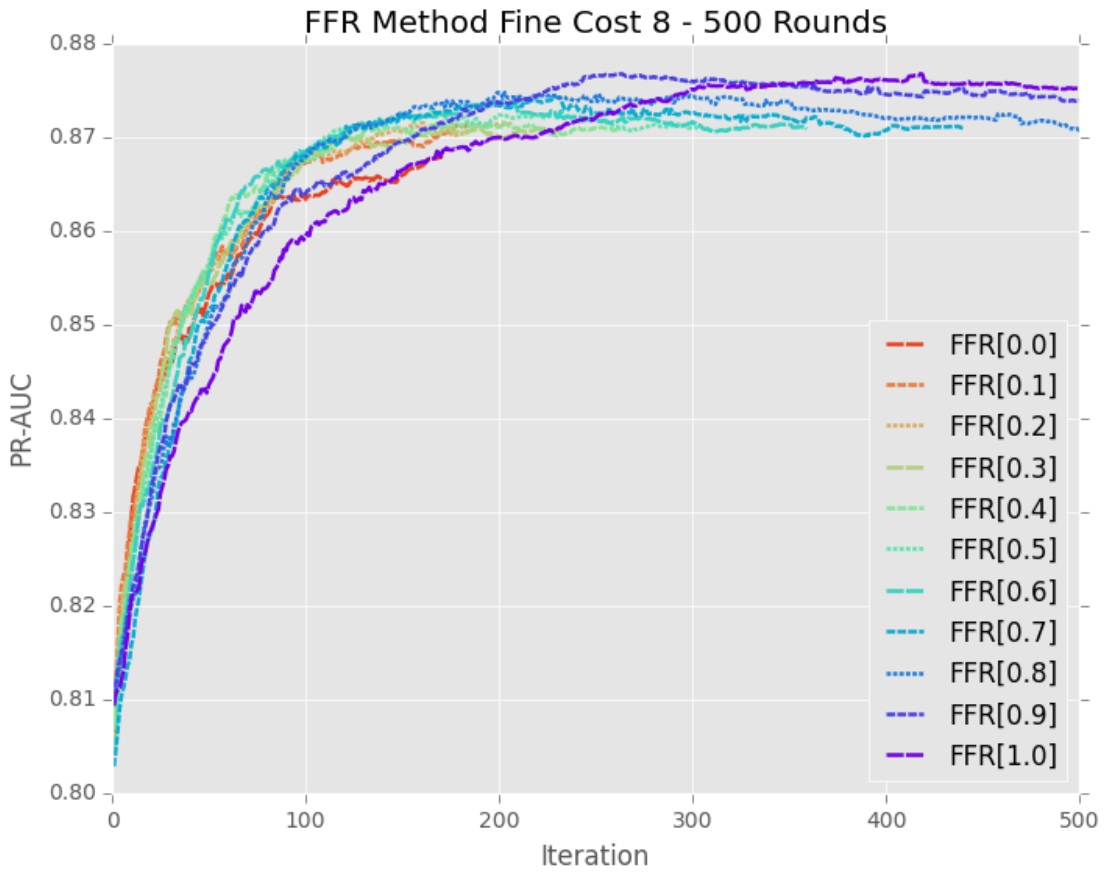


Figure 5.16: This shows the iterations continuing through round 500, the curves with the higher fine rates eventually settle to the same end point that the curves with the high rates of coarse labels purchased achieved at previous iterations.

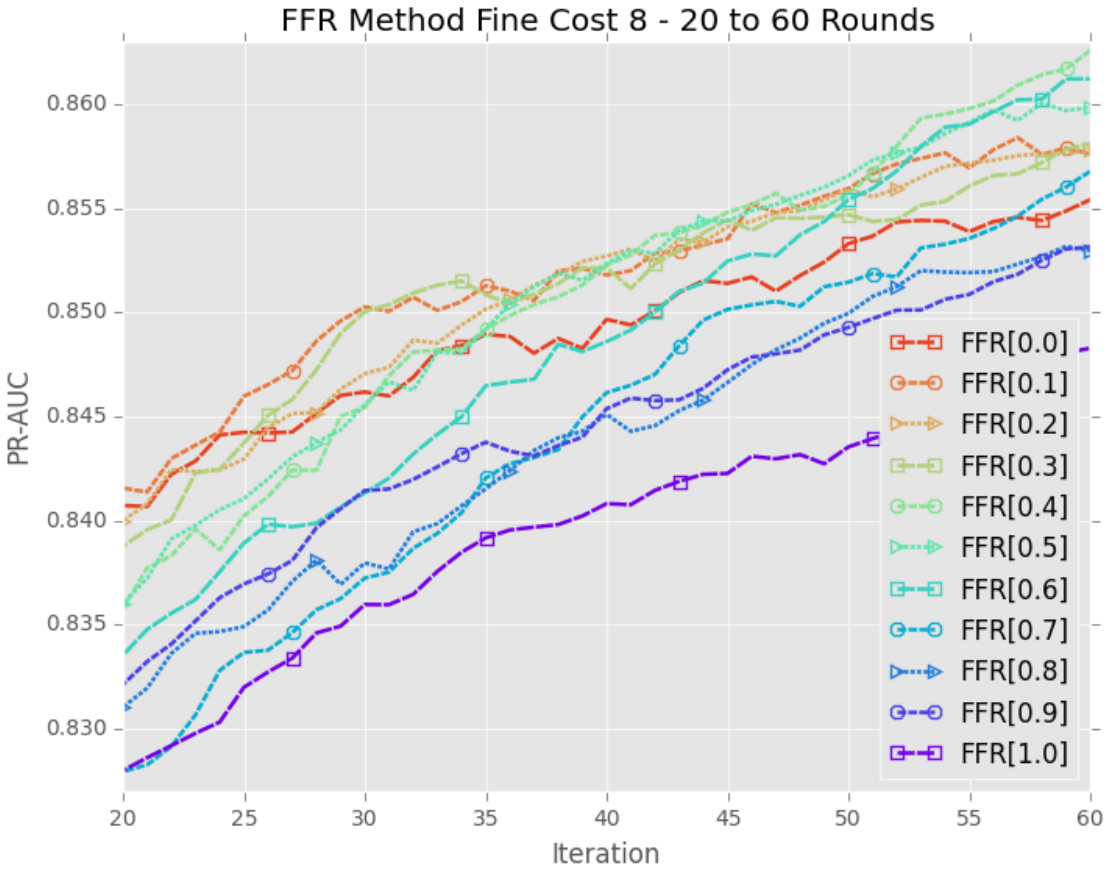


Figure 5.17: The fine cost 8 curves shown expanding the rounds 20-60. If a round budget of 40 occurs than the recommended FFR would be 0.2

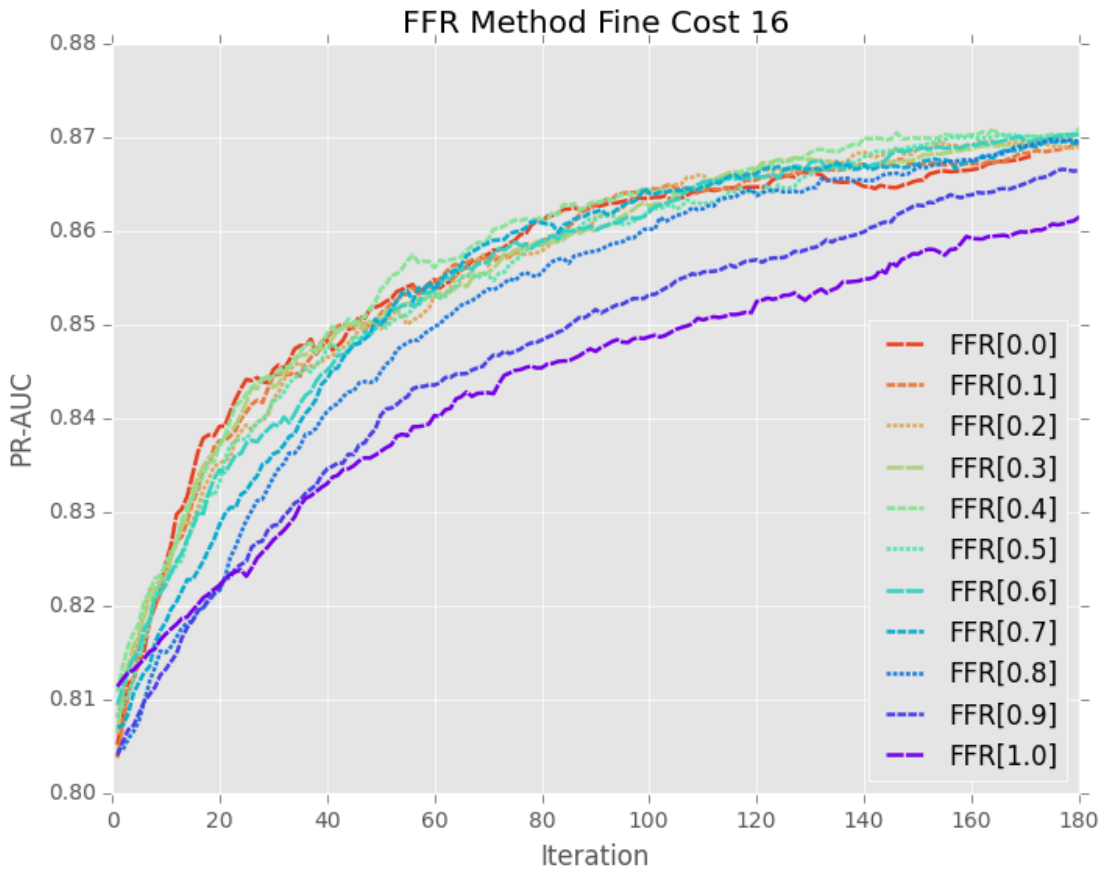


Figure 5.18: The fine cost is increased to 16. The cost is to high for the fine label advantage to offset the decreased number of instances purchased.

Chapter 6

Conclusions and Future Work

The protein data set demonstrated that fine-grained labels can be used to improve the coarse-grained classifier performance. Both SVM and Logit classifiers show an advantage of around 0.005 in average PR curve AUC and around 0.001 in ROC curve AUC. Experiments comparing active and passive learning for both fine and coarse classifiers are performed, the results shown in section 5.2 demonstrate a prominent advantage for active fine with the Logit classifier. Furthermore HAL is implemented and applied to the protein dataset for various FFR proportions and fine label costs. After discussions with Cui et. al's group at UNL [1] the fine label estimated cost is around 10. This correlates to the FFR experiment with fine cost 8, which shows that for a budget of around 20-60 rounds a FFR strategy of 0.2 would have the highest performance, see *Figure 5.17*.

Future work is to apply the BANDIT approach to the protein dataset in order to achieve an optimal strategy for selecting the FFR proportion vector throughout all rounds, see section 2.3. The active over-labeling approach could be applied to other datasets with more complex hierarchical label trees; datasets derived from Gene Ontology research could be investigated [30].

Bibliography

[1] J. Z. Juan Cui, Kevin Chiang, "Prediction of nuclear and locally encoded mitochondrion." Lincoln, NE: Nebraska Gateway to Nutrigenomics 6th Annual Retreat, June 9 2014. [Online]. Available: http://cehs.unl.edu/nutrigenomics/nebraska-gateway-nutrigenomics-6th-annual-retreat/ 1, 1.3, 1.1, 2.2, 5.3, 6

[2] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997. 1.1, 2.1

[3] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122. 1.1

[4] A. Ng, "Machine Learning by Standford University," https://www.coursera.org/learn/machine-learning/home/welcome, 2016, accessed: 2016-12-5. 1.1

[5] 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*, vol. 12, pp. 2825–2830, 2011. 1.1, 1.1, 1.1, 4.1, 4.1.1, 4.1.2, 4.1.3, 4.1.9

[6] E. Alpaydin, *Introduction to Machine Learning (Adaptive Computation and Machine Learning series)*, 2nd ed. Cambridge, Massachusetts: The MIT Press, 2009. [Online]. Available: https://amzn.com/026201243X 1.1

[7] D. Cotter, P. Guda, E. Fahy, and S. Subramaniam, "Mitoproteome: mitochondrial protein sequence database and annotation system," *Nucleic Acids Research*, vol. 32, no. suppl1, p. D463, 2004. [Online]. Available: +http://dx.doi.org/10.1093/nar/gkh048 1.3

[8] "Activities at the universal protein resource (uniprot)," *Nucleic Acids Research*, vol. 42, no. D1, p. D191, 2014. [Online]. Available: http://dx.doi.org/10.1093/nar/gkt1140 1.3

[9] Z. R. Li, H. H. Lin, L. Y. Han, L. Jiang, X. Chen, and Y. Z. Chen, "Profeat: a web server for computing structural and physicochemical features of proteins and peptides from amino acid sequence," *Nucleic Acids Research*, vol. 34, no. Web Server issue, pp. W32–W37, 07 2006. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1538821/ 1.1

[10] J. Cui, L. Y. Han, H. Li, C. Y. Ung, Z. Q. Tang, C. J. Zheng, Z. W. Cao, and Y. Z. Chen, "Computer prediction of allergen proteins from sequence-derived protein structural and physicochemical properties," *Molecular Immunology*, vol. 44, no. 4, pp. 514 – 520, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0161589006000368 1.1

[11] J. M. Brown and A. J. Giaccia, "The unique physiology of solid tumors: Opportunities (and problems) for cancer therapy," *Cancer Research*, vol. 58, no. 7, pp. 1408–1416, 1998. [Online]. Available: http://cancerres.aacrjournals.org/content/58/7/1408 1.1

[12] L. Kll, A. Krogh, and E. L. L. Sonnhammer, "An hmm posterior decoder for sequence feature prediction that includes homology information," *Bioinformatics*, vol. 21, no. suppl-1, p. i251, 2005. [Online]. Available: +http://dx.doi.org/10.1093/bioinformatics/bti1014 1.1

[13] F. Eisenhaber, C. Frmmel, and P. Argos, "Prediction of secondary structural content of proteins from their amino acid composition alone. ii. the paradox with secondary structural class," *Proteins: Structure, Function, and Bioinformatics*, vol. 25, no. 2, pp. 169–179, 1996. [Online]. Available: http://dx.doi.org/10.1002/(SICI)1097-0134(199606)25:2⟨169::AID-PROT3⟩3.0.CO;2-D 1.1

[14] J. D. Bendtsen, H. Nielsen, G. von Heijne, and S. Brunak, "Improved prediction of signal peptides: Signalp 3.0," *Journal of Molecular Biology*, vol. 340, no. 4, pp. 783 – 795, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022283604005972 1.1

[15] A. G. Garrow, A. Agnew, and D. R. Westhead, "Tmb-hunt: a web server to screen sequence sets for transmembrane -barrel proteins," *Nucleic Acids Research*, vol. 33, no. suppl2, p. W188, 2005. [Online]. Available: http://dx.doi.org/10.1093/nar/gki384 1.1

[16] K. Julenius, A. Mlgaard, R. Gupta, and S. Brunak, "Prediction, conservation analysis, and structural characterization of mammalian mucin-type o-glycosylation sites," *Glycobiology*, vol. 15, no. 2, p. 153, 2005. [Online]. Available: http://dx.doi.org/10.1093/glycob/cwh151 1.1

[17] J. D. Bendtsen, H. Nielsen, D. Widdick, T. Palmer, and S. Brunak, "Prediction of twin-arginine signal peptides," *BMC bioinformatics*, vol. 6, no. 1, p. 167, 2005. 1.1

[18] Y. Mo, S. D. Scott, and D. Downey, "Learning hierarchically decomposable concepts with active over-labeling," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Dec 2016, pp. 340–349. 1.4, 2.1, 1, 2.3, 3, 3.1, 3.1

[19] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "RCV1: A New Benchmark Collection for Text Categorization Research," *Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004. 2.1, 2.1, 3.1

[20] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2–3, pp. 235–256, 2002. 2.3

[21] A. McCallum, R. Rosenfeld, T. M. Mitchell, and A. Y. Ng, "Improving text classification by shrinkage in a hierarchy of classes," in *Proceedings of the Fifteenth International Conference on Machine Learning*, ser. ICML '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 359–367. [Online]. Available: http://dl.acm.org/citation.cfm?id=645527.657461 3

[22] W. Jiang and Z. W. Ras, "Multi-label automatic indexing of music by cascade classifiers," *Web Intelli. and Agent Sys.*, vol. 11, no. 2, pp. 149–170, Apr. 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2590084.2590088 3

[23] N. Rubens, D. Kaplan, and M. Sugiyama, "Active learning in recommender systems," *Recommender Systems Handbook*, pp. 1–31, 2011. [Online]. Available: http://link.springer.com/chapter/10.1007/978-0-387-85820-3_23 3

[24] A. Merialdo, "Improving Collaborative Filtering For New-Users By Smart Object Selection," *In Proceedings of International Conference on Media Features (ICMF)*, May 2001. [Online]. Available: http://www.eurecom.fr/publication/670https://www.eurecom.fr/fr/publication/670/download/mm-kohrar-010508.pdf 3

[25] T. Hofmann, "Collaborative filtering via gaussian probabilistic latent semantic analysis," in *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval - SIGIR '03*. New York, New York, USA: ACM Press, Jul. 2003, p. 259. [Online]. Available: http://dl.acm.org/citation.cfm?id=860435.860483 3

[26] G. Schohn and D. Cohn, "Less is more: Active learning with support vector machines," *ICML*, pp. 839–846, Jun. 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=645529.657802http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.6090&rep=rep1&type=pdf 3

[27] S. Dasgupta and D. Hsu, "Hierarchical sampling for active learning," *Proceedings of the 25th international conference on Machine learning - ICML '08*, pp. 208–215, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1390156.1390183 3

[28] C. Symons, N. Samatova, R. Krishnamurthy, B. Park, T. Umar, D. Buttler, T. Critchlow, and D. Hysom, "Multi-Criterion Active Learning in Conditional Random Fields," *2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, pp. 323–331, Nov. 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4031915http://dl.acm.org/citation.cfm?id=1190614.1191040 3

[29] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 233–240. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143874 5.2.1

[30] GO Consortium, "The Gene Ontology," 2014. [Online]. Available: geneontology.org 6