HIERACHICAL ACTIVE LEARNING BIOINFORMATICS APPLICATION

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

HIERACHICAL ACTIVE LEARNING BIOINFORMATICS APPLICATION

James D. Duin, M.S.

University of Nebraska, 2017

Adviser: Stephen Scott

This study investigates an application of the Support Vector Machine and Logistic Regression machine learning algorithms to a protein dataset labeled according to a proteins location of origin in a cell. The dataset is labeled according to a hierarchical scheme, at the root level the protein is 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 classifer at the highest level in the hierarchy according to the Precision-Recall curves area under the curve. Furthermore, new approach in the Active Learning setting termed *active over-labeling* is applied to this dataset. The approach solicits labels at a finer level of grainularity than the target concept. We show that purchasing fine grained labels in each round of active learning produces a higher performing classifier than purchasing coarse labels, and in both cases purchasing labels actively by selecting only the most uncertain labels outperforms purchasing labels passively, i.e. at random. The fine grained labels also incur a higher cost than coarse grained labels for this dataset, so multiple cost ratios ran and an optimal Fixed Fine ratio purchasing strategy is determined for each fine cost.

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 guidance in selecting this research topic, Yugi Mo for his work in developing the HAL methodology, and Dr. Douglas Downey for his work on this topic. I would like to thank Juan Cui, Jiang Shu, Kevin Chiang for assistance accessing and understanding the protein dataset that is the subject of the paper.

Contents

Chapter 1

Introduction

1.1 Machine Learning

Machine Learning 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 [1]. In the context of this paper, the machine learning algorithms that are used include a support vector machine (SVM) implementation, and a Logistic Regression (LogReg) implementation by the sci-kit learn python library [2]. The performance measure is the classification of protein instances according to a label, for example, originated in the mitochondria or not. The experience is the number of training instances in the training set. The dataset is initially partitioned into training and test sets. The algorithm improves its classifier structure based on the features of each instance by iterating through all instances in the training set. When training has been completed, the classifier is then tested on the test set and the number of instances that the classifier correctly or incorrectly labels determines its accuracy, precision and recall scores. In our protein dataset there are 20,098 proteins with 449 features each relating to their structure. In our experiments an SVM classifier is applied to the dataset with the goal of achieving high precision scores for the label mitochondrion, that is, if the protein originates in the mitochondria or not.

1.2 Hierarchical Bioinformatics Data Set

The protein dataset is labeled according to where it originates in the cell. At the root is mitochondrion, then there is the sub level labels for if its native to the mitochondria or if it has a separate target compartment specification. The complete tree along with the number of instances belonging to the each label is included in *Figure 1.1* .

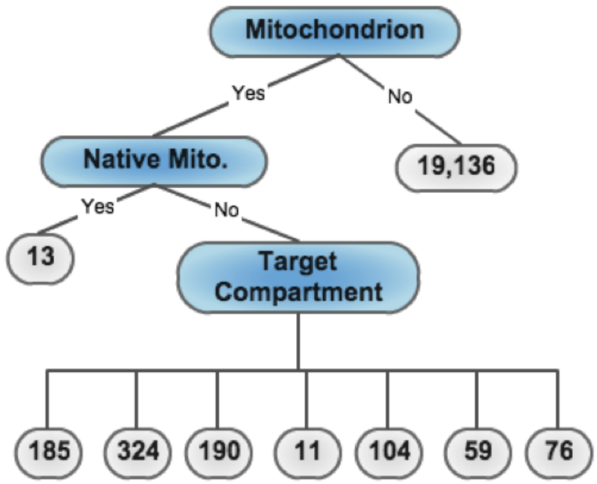


Figure 1.1: The protein dataset is labeled according to where it originates in the cell. At the root is mitochondrion, then there is the sub level labels for if its native to the mitochondria or if it has a separate target compartment specification. The complete tree along with the number of instances belonging to the each label is included in this figure.

1.3 Coarse Grained vs Fine Grained Trade Off

The classifier that does not take advantage of any of the fine grained labels works off of the root labels for each instance and does not train separate classifiers for the fine grained labels. This classifier is referred to as the coarse grained classifier. The classifier that does use fine grained labels, and trains a separate classifier for each label, then combines them to generate a root level label is referred to as the fine grained classifier. It can be demonstrated through a dummy 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 course 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 dummy dataset pictured. It is the intention of this study to demonstrate that 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.

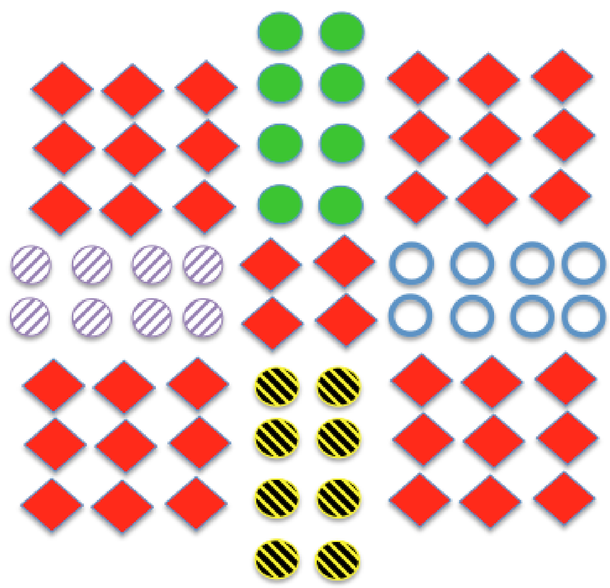


Figure 1.2: Demonstration of a dataset that would benefit from multiple fine grained learners for each circle type. In order for the coarse grain learner to have high recall, precision must be scarified and a large amount of false positives returned. By combining fine grained classifiers the same level of recall can be achieved with a higher level of precision because none of the false positive diamonds will be returned

Chapter 2

Background and Related Work

2.1 Active Learning

Active Learning relates to the coarse grained vs fine grained tradeoff because it is reasonable to assume that fine grained labels may not be as readily available as coarse grained labels, and thus have a higher cost. An active learning approach is used to determine how many fine grained labels to purchase in order to minimize the total cost to train the algorithm an maximize the precision and recall scores. The following equations for precision, recall, and a weighted F score are shown below in equations 1 through 3.

$Precisioneqn$

$Recalleqn$

$F0_5eqn$

The goal in an active learning approach is to maximize the F measure where equals 0.5 [2]. The F-0.5 measure gives more weight to precision, as opposed to recall, so it gives incentive to purchase enough fine grained labels to increase the F-0.5 measure. The coarse grained labels will cost less than fine grained labels, but the increase in the F-0.5 measure justifies the increase in cost up to a certain point. The F-0.5 measure is used in the results section of this paper. ... That's why we use PRauc in the results. Describe some of the other papers that Yugi cited.

2.2 Hierarchical Active Learning

The Hierarchical Active Learning algorithm (HAL) is shown diagrammatically in Figure 3. Multiple fine grained classifiers are trained at each level of the Hierarchy of the dataset. Queries to the oracle are performed purchase the most cost effective labels to add to the training sets of the classifiers. The active learning cycle continues until a cost budget has been reached. The benefit of an active learning approach is to maximize the F-0.5 measure for a given cost budget. It was the goal of this study to apply the HAL algorithm to the protein dataset, however this is not achieved at this time. An existing application of HAL is briefly discussed in the following section.

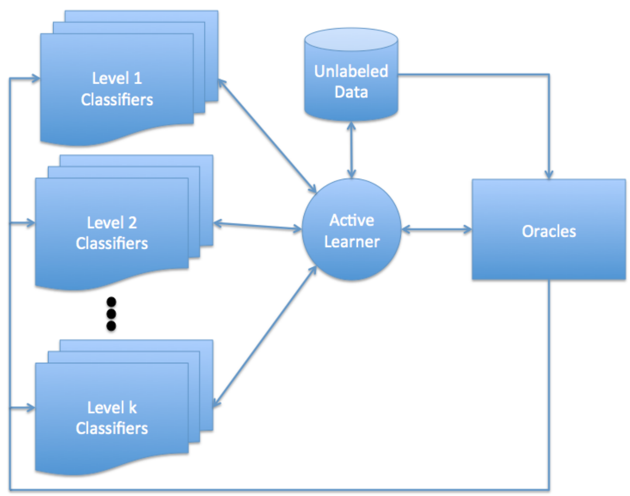


Figure 2.1: Diagram of HAL approach

2.3 Application to Dispatch Dataset

HAL was applied to a Dispatch dataset. This dataset contains 375,026 manually labeled hierarchical names across 1,384 newspaper articles [2]. This is a clear example where fine grained labels have a higher cost since it is easier for a person to manually determine if the article pertains to an organization or not, rather than if it pertains to a railroad or a zoo, which would be sub labels of the organization root. The first analysis step was to determine that the F-0.5 measure is increased by using fine grained classifiers. The results are shown in Figure 4. The highest F-0.5 measure for a given iteration of purchases of training instances is obtained by using the active learning approach with all fine-grained labels. The passive learning curves were generated by selecting batches of instances randomly rather than querying the oracle for a specific label type that offers that most gain in classifier accuracy. The active learning curves did take advantage of querying for specific labels in order to maximize gain in classifier accuracy. ... add other data sets.

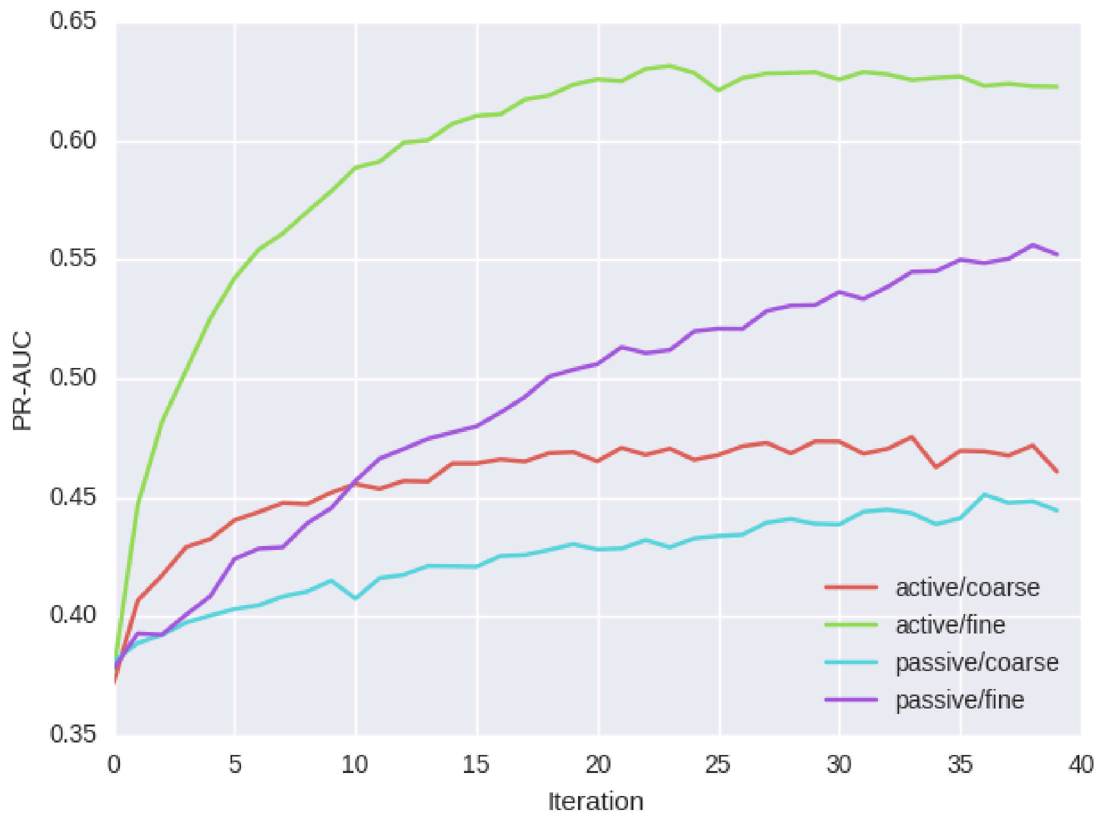


Figure 2.2: Application of HAL demonstrating the benefit of Actively selecting the type of labels to purchase for instances rather than randomly selecting labels to purchase, as in the Passive curves.

The next analysis step is to apply a given ratio of fine grained vs coarse grained labels to purchase at each batch request to the oracle. The results of varying the percentage of fine grained labels purchased are shown in Figure 5. The figure shows that even a small amount of fine grained labels purchased, that is, 20perc provides a significant increase in the F-0.5 measure for a given iteration. ... pr auc results. Add explanation of Bandit approach and results.

Chapter 3

Experimental Setup

3.1 Training and Testing Coarse Grain and Fine Grain Classifiers

The bioinformatics dataset is composed of 9 classes as shown in *Figure 1.1* on *page 2*. 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 mitochondrian instances are treated as negative or 0 labeled. The totals for each class type is shown in *Table 3.1a* . Throughout this experiment a 10 folds cross validation strategy is used, an example partitioning in shown in *Table 3.1b* .

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

Table 3.1: 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 positve instance.

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 3.2*.

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

Table 3.2: Example of totals for the Train and Test corresponding to when the first fold is held out to be the test set.

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 ran rapidly and for the class weight parameter to be tuned for various round sizes. The reduced subset contains a randomly chosen group of approximately $1/5^{th}$ of the negatives. The class totals and example partitioning for the reduced subset is shown in *Table 3.3*.

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 3.3: The subset of instances used for tuning classifier paramters contains approximately $1/5^{th}$ and retains all postive instances.

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

Table 3.4: 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.

I tried using SVM. Throughout this project I used the python library sci-kit learn [2]. The support vector machine implemented by this library has the following default parameters. SVC C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache-size=200, class-weight=None, verbose=False, max-iter=-1, decision-function-shape=None, random-state=None.

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 3.5: SVMDef result stats

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 3.6: SVMDef conf matrix

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)

Table 3.7: SVMDef

3.1.1 Varied SVM Scaling Methods

I tried different scaling methods (min max scaler, std scaler), I settled on std scaler. [2]

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 3.8: SVMMinMax

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 3.9: SVMNorm

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)

Table 3.10: SVMStandard

3.1.2 Varied SVM Kernels

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)

Table 3.11: RbfOrig

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 3.12: Linear

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 3.13: polyDeg3

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 3.14: polyDeg6

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 3.15: sigmoid

3.1.3 Varied SVM Feature Selection

I tried different feature select measures, I settled on 75 perc feature select.

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 3.16: SVMSel25

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 3.17: SVMSel50

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)

Table 3.18: SVMSel75

3.1.4 Varied SVM Decision Function Shape

I tried different different C costs, kernels, decision function shape, gamma, tolerance settled on classif = svm.SVC(C=1.0, kernel='rbf',decisionfunctionshape='ovo',gamma=0.0025, tol=0.00001). 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 wiched weights each class by the number of instances it has in the train set. Tolerance default is 0.001.

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.913	0.883	0.878	0.621	(372.1 / 48.0)	(10.6 / 48.1)
fine	0.880	0.847	0.809	0.089	(382.7 / 91.6)	(0.0 / 4.5)

Table 3.19: SVM-ovr

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)

Table 3.20: SVM-ovo

3.1.5 Varied Logistic Regression Scaling

I left the class weight as balanced for this part, the results did not show much advantage for using the fine grained classifier.

next I tried using a Logistic regression classifier.

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 3.21: LogRegDef

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 3.22: LogRegStandard

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 3.23: LogRegNorm

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)

Table 3.24: LogRegMinMax

3.1.6 Varied Logistic Regression Feature Selection

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 3.25: LogRegSel25

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 3.26: LogRegSel50

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 3.27: LogRegSel75

3.1.7 Varied Logistic Regression Postive Class Weight and Cost

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 3.28: LogRegWtOrig-C1

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 3.29: LogRegWtOrig-Cp1

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 3.30: LogRegWtOrig-C10

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 3.31: LogRegWt10-C1

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 3.32: LogRegWt10-Cp1

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 3.33: LogRegWt10-C10

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 3.34: LogRegWt7p5-C1

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 3.35: LogRegWt7p5-Cp1

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)

Table 3.36: LogRegWt7p5-C10

3.1.8 Varied Logistic Regression Tune Fine Class Weights

3.1.8.1 Fine Tune Class 1 Weights

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 3.37: LogRegCls1-Wt1

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 3.38: LogRegCls1-Wtp5

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 3.39: LogRegCls1-Wt3: this won!

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)

Table 3.40: LogRegCls1-Wt5

3.1.8.2 Fine Tune Class 2 Weights

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 3.41: LogRegCls2-Wt1: this won!

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 3.42: LogRegCls2-Wtp5

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)

Table 3.43: LogRegCls2-Wt1p5

3.1.8.3 Fine Tune Class 3 Weights

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 3.44: LogRegCls3-Wt1: this won!

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 3.45: LogRegCls3-Wtp5

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)

Table 3.46: LogRegCls3-Wt1p5

3.1.8.4 Fine Tune Class 4 Weights

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 3.47: LogRegCls4-Wt1

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 3.48: LogRegCls4-Wtp5

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 3.49: LogRegCls4-Wt1p5: this won!

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)

Table 3.50: LogRegCls4-Wt2

3.1.8.5 Fine Tune Class 5 Weights

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 3.51: LogRegCls5-Wt1

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 3.52: LogRegCls5-Wtp5

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 3.53: LogRegCls5-Wt1p5

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 3.54: LogRegCls5-Wt5

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 3.55: LogRegCls5-Wt10: this won!

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)

Table 3.56: LogRegCls5-Wt20

3.1.8.6 Fine Tune Class 6 Weights

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 3.57: LogRegCls6-Wt1

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 3.58: LogRegCls6-Wtp5

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 3.59: LogRegCls6-Wt2: this wins!

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)

Table 3.60: LogRegCls6-Wt3

3.1.8.7 Fine Tune Class 7 Weights

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 3.61: LogRegCls7-Wt1

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 3.62: LogRegCls7-Wtp5

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 3.63: LogRegCls7-Wt3: this won!

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)

Table 3.64: LogRegCls7-Wt5

3.1.8.8 Fine Tune Class 8 Weights

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 3.65: LogRegCls8-Wt1: this won!

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 3.66: LogRegCls8-Wtp5

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)

Table 3.67: LogRegCls8-Wt1p5

3.1.9 Varied Logistic Regression Threshold

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 3.68: LogRegAftFineTune

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 3.69: LogRegOrig-0001

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 3.70: LogReg-00001 - this won!

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 3.71: LogReg-000001

3.1.10 Varied Sample Weight On Test Set and Dropping Intermediate ROC Curve Values

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 3.72: LogReg-NoSW

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 3.73: LogReg-DropFalse

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 3.74: LogReg-NoSW-DropFalse

3.1.11 Varied Logistic Regression Positive Class Weight For Full Dataset

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 3.75: LogRegAllOrig-Wt20p887

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 3.76: LogRegAll-Wt23: this won!

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)

Table 3.77: LogRegAll-Wt25

3.1.12 Varied SVM Gamma

title	pr	roc	acc	f1	conf (tn/fn)	conf (fp/tp)
coarse	0.901	0.874	0.847	0.653	(336.6 / 27.2)	(46.1 / 68.9)
fine	0.896	0.864	0.871	0.597	(371.1 / 50.2)	(11.6 / 45.9)

Table 3.78: SVM-OrigWithFtune

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 3.79: SVM-C1-Gp0029674

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 3.80: SVM-C2-Gp0029674

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 3.81: SVM-Cp1-Gp0029674

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 3.82: SVM-Cp05-Gp0029674

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 3.83: SVM-Cp15-Gp0029674: this won!

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 3.84: SVM-Cp2-Gp0029674

3.1.13 Varied SVM Cost

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 3.85: SVM-Cp15-Gp002: this won!

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)

Table 3.86: SVM-Cp15-Gp001

I tried different scaling methods, min max scaler, std scaler, I settled on min max scaler.

I tried different feature select measures, decided to use all of the features.

I tried different C costs, tolerances, and class weights. I settled on C=0.1, tol = 0.00001, and weight equal to the balanced, adjusted via a scaling line.

I also further tuned the fine grained classifiers starting from the initial scaling from the line an then multiplying that by a ratio. I got this vector of ratios [0.87, 0.4, 0.78, 0.65, 3.48, 0.78, 1.74, 0.87]

Chapter 4

Results and Analysis

4.1 Passive SVM Rbf kernel vs Logistic Reg

This shows the advantage to fine grained labels to justify the experiment.

coarse-pr	fine-pr	coarse-roc	fine-roc	coarse-acc	fine-acc	coarse-f1	fine-f1
0.898	0.905	0.905	0.899	0.767	0.912	0.259	0.404
0.870	0.871	0.846	0.852	0.803	0.918	0.272	0.406
0.897	0.908	0.895	0.902	0.793	0.919	0.287	0.449
0.864	0.860	0.852	0.852	0.778	0.908	0.256	0.391
0.855	0.864	0.859	0.857	0.795	0.920	0.269	0.423
0.867	0.863	0.874	0.864	0.785	0.913	0.263	0.411
0.871	0.882	0.873	0.879	0.784	0.910	0.269	0.404
0.835	0.842	0.843	0.841	0.794	0.910	0.258	0.357
0.869	0.878	0.868	0.875	0.785	0.914	0.265	0.418
0.873	0.873	0.891	0.890	0.786	0.906	0.279	0.365
avg 0.870	avg 0.875	avg 0.871	avg 0.871	avg 0.787	avg 0.913	avg 0.268	avg 0.403

Table 4.1: Here are the results for the logistic regression passive 10 folds.

coarse-tn	fine-tn	coarse-fp	fine-fp	coarse-fn	fine-fn	coarse-tp	fine-tp
1460	1773	454	141	14	36	82	60
1540	1790	374	124	22	40	74	56
1509	1782	405	132	12	30	84	66
1486	1767	428	147	19	37	77	59
1521	1790	393	124	20	37	76	59
1501	1774	413	140	19	35	77	61
1496	1769	417	144	17	36	80	61
1524	1780	389	133	25	47	72	50
1498	1773	415	140	17	33	78	62
1497	1767	416	146	13	42	83	54
avg 1503.2	avg 1776.5	avg 410.4	avg 137.1	avg 17.8	avg 37.3	avg 78.3	avg 58.8

Table 4.2: Here are the results for the logistic regression confusion matrices. The main source of the advantage for fine is from the decreased amount of false negatives.

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 4.3: LogRegAll-Wt23

coarse-pr	fine-pr	coarse-roc	fine-roc	coarse-acc	fine-acc	coarse-f1	fine-f1
0.894	0.914	0.890	0.900	0.862	0.944	0.351	0.525
0.896	0.902	0.886	0.888	0.870	0.948	0.359	0.514
0.885	0.889	0.856	0.859	0.865	0.941	0.333	0.482
0.893	0.893	0.881	0.874	0.859	0.942	0.331	0.496
0.886	0.884	0.872	0.872	0.865	0.932	0.340	0.445
0.879	0.887	0.863	0.870	0.868	0.943	0.342	0.472
0.902	0.899	0.892	0.886	0.868	0.945	0.342	0.488
0.879	0.896	0.859	0.875	0.860	0.943	0.332	0.491
0.911	0.913	0.904	0.903	0.871	0.943	0.360	0.482
0.891	0.901	0.898	0.890	0.874	0.940	0.375	0.452
avg 0.892	avg 0.898	avg 0.880	avg 0.882	avg 0.866	avg 0.942	avg 0.347	avg 0.485

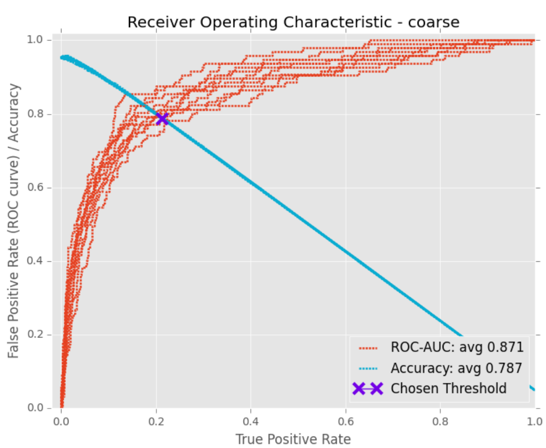
Table 4.4: Here are the results for the SVM passive 10 folds.

coarse-tn	fine-tn	coarse-fp	fine-fp	coarse-fn	fine-fn	coarse-tp	fine-tp
1658	1836	256	78	21	34	75	62
1676	1851	238	63	23	41	73	55
1670	1837	244	77	28	41	68	55
1657	1837	257	77	26	39	70	57
1668	1818	246	96	26	41	70	55
1676	1845	238	69	27	45	69	51
1676	1846	237	67	28	44	69	53
1658	1841	255	72	27	42	70	55
1676	1841	237	72	22	42	73	53
1680	1838	233	75	20	46	76	50
avg 1669.5	avg 1839.0	avg 244.1	avg 74.6	avg 24.8	avg 41.5	avg 71.3	avg 54.6

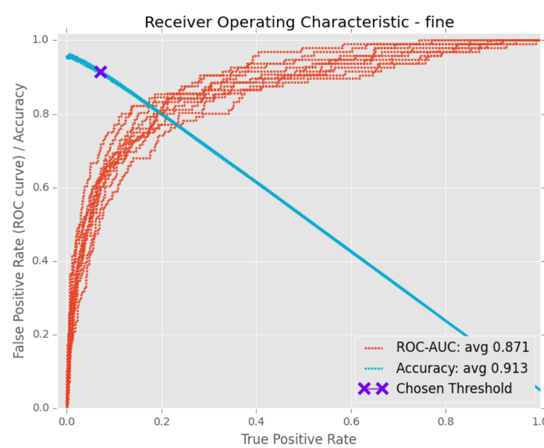
Table 4.5: Here are the results for the SVM confusion matrices. Here the fine returns less false negatives than the coarse, but not as many true positives compared to coarse.

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)

Table 4.6: SVM-All

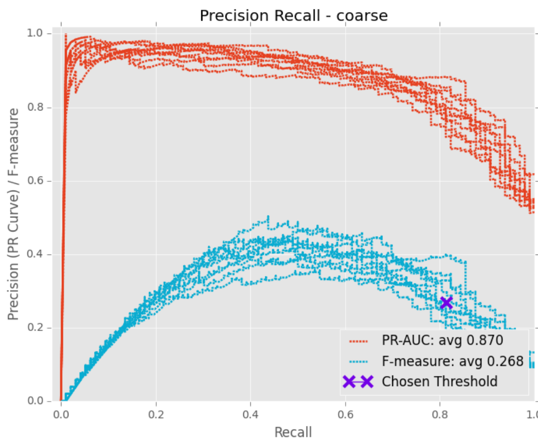


(a) Log Reg ROC Curves - coarse

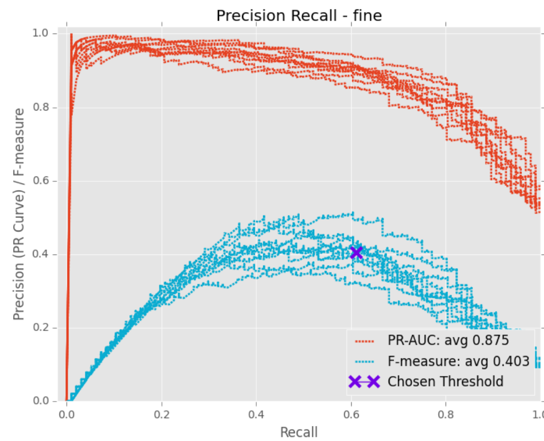


(b) Log Reg ROC Curves - fine

Figure 4.1: Fine has a higher accuracy than coarse at the default threshold.

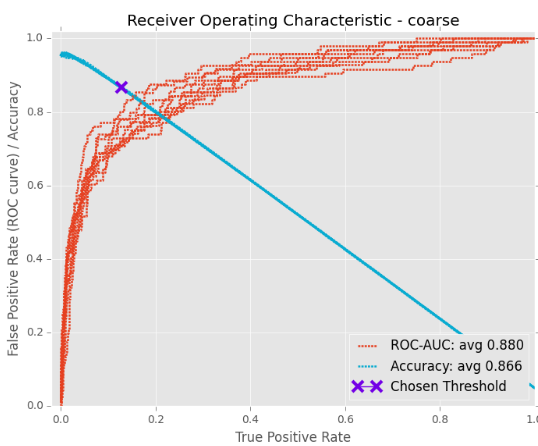


(a) Log Reg Pr Curves - coarse

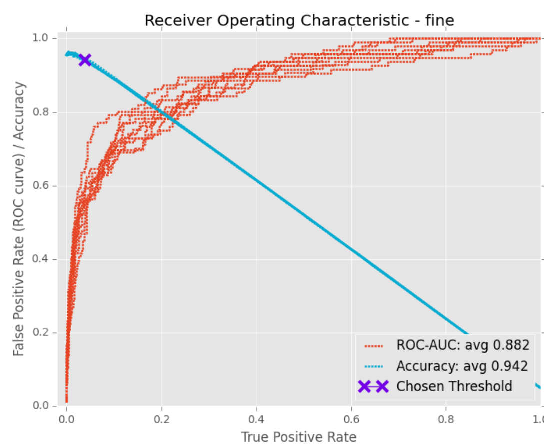


(b) Log Reg Pr Curves - fine

Figure 4.2: The fine threshold occurs at a point on the pr-curve associated with a higher f-measure than the coarse curves.

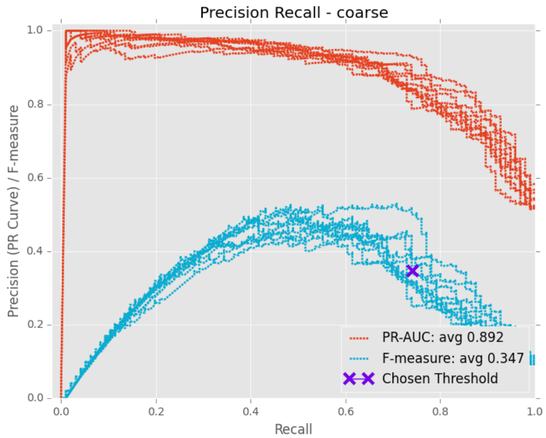


(a) SVM ROC Curves - coarse

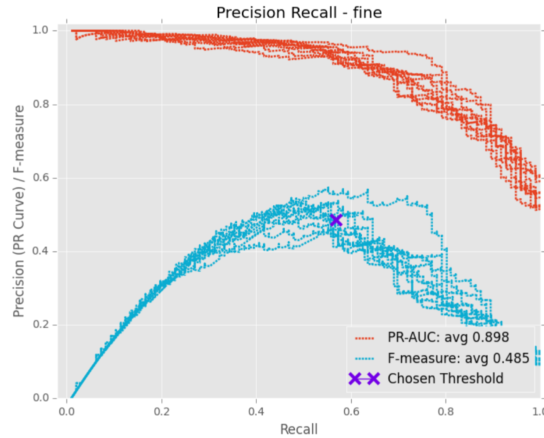


(b) SVM ROC Curves - fine

Figure 4.3: SVM results are similar between coarse and fine.



(a) SVM Pr Curves - coarse



(b) SVM Pr Curves - fine

Figure 4.4: SVM results for PR-curves and F-measure have coarse and fine picking different parts of the curves for their respective thresholds, coarse f1 avg is slightly higher at 0.468 compared to 0.457 for fine.

4.2 Active vs Passive curves

The following plots were obtained with a round batch size of 100 and a starter set of 1040 instances out of the total 2098 instances. The plots are the average of 10 folds, for each fold a test set of 2010 containing representatives of each class was held out, out of the remaining 18088, the starter set was selected which again contained representatives of each class. Coarse and fine classifiers share the same starter set. During each round coarse and fine classifiers are trained on their corresponding sets, metrics are outputted on the held out test set, then confidence estimates are ran on the remaining eligible instances. Eligible instances are kept in separate sets for coarse and fine, 100 of the most uncertain instances are removed from each eligible set and added to its corresponding coarse or fine set to be trained on for the next round.

4.2.1 Plots for Logistic Regression Active vs Passive curves

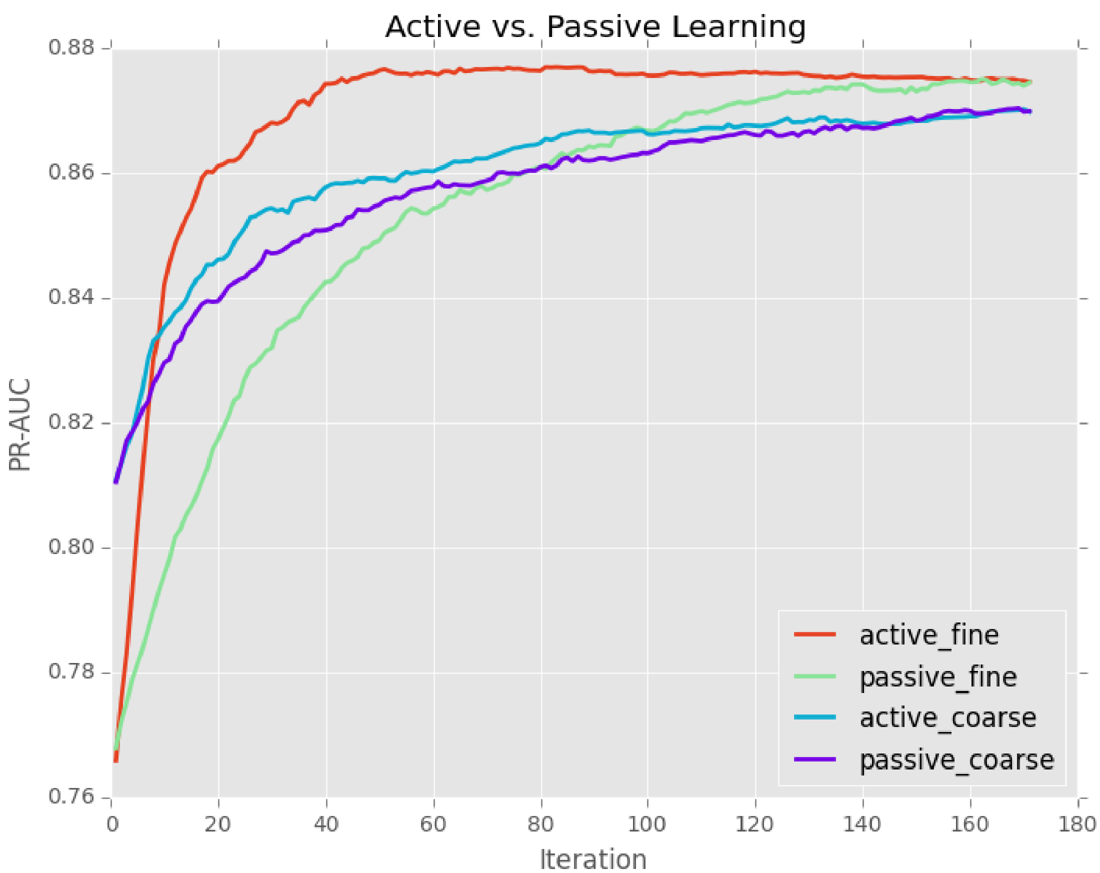


Figure 4.5: The PR AUC curves for rounds with the Logistic Regression classifier conforms to expectations, with active-fine having the highest performance. Active-coarse outperforms passive-coarse. Passive-fine doesn't outperform the coarse classifiers until rnd 100.

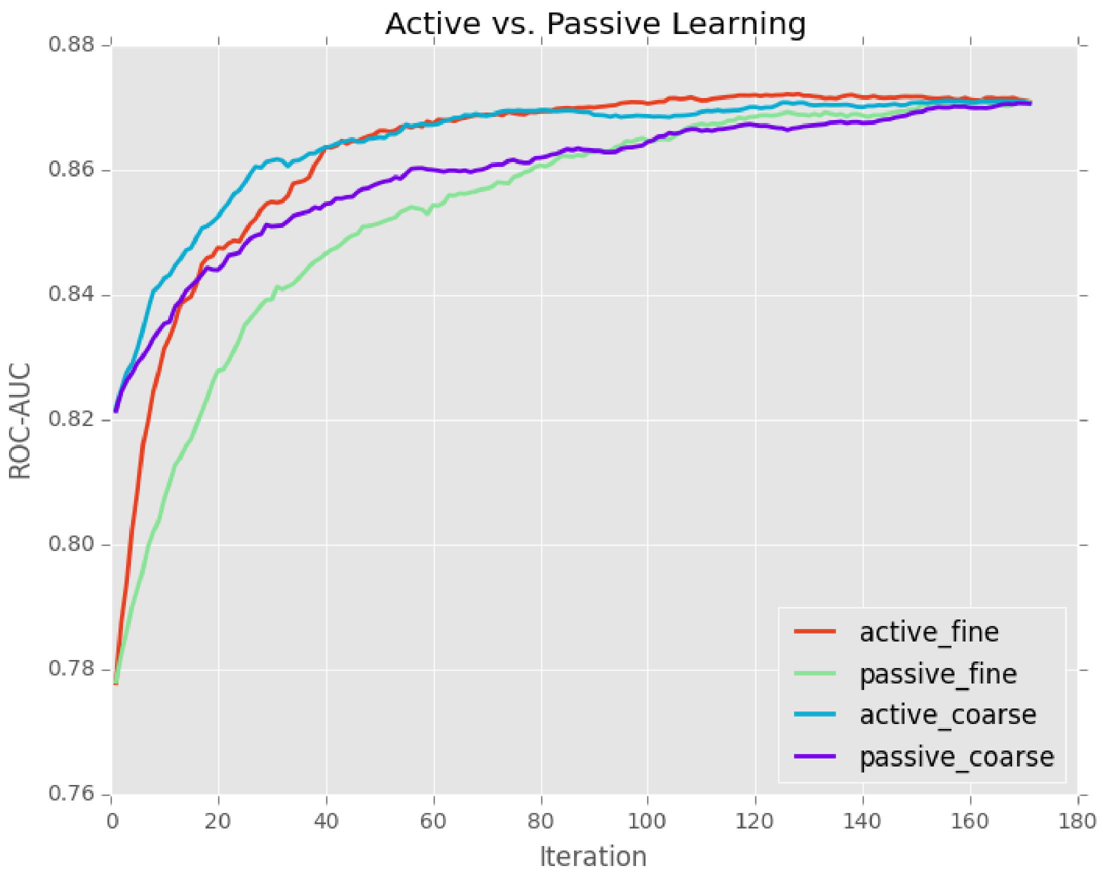


Figure 4.6: The ROC AUC curves for rounds with the Logistic Regression classifier. The active curves beat out the passive curves for both coarse and fine. Coarse roc starts with an advantage over fine as in the PR curves. Both converge to the same rate after roc auc level after 80.

4.2.2 Plots for SVM Active vs Passive curves

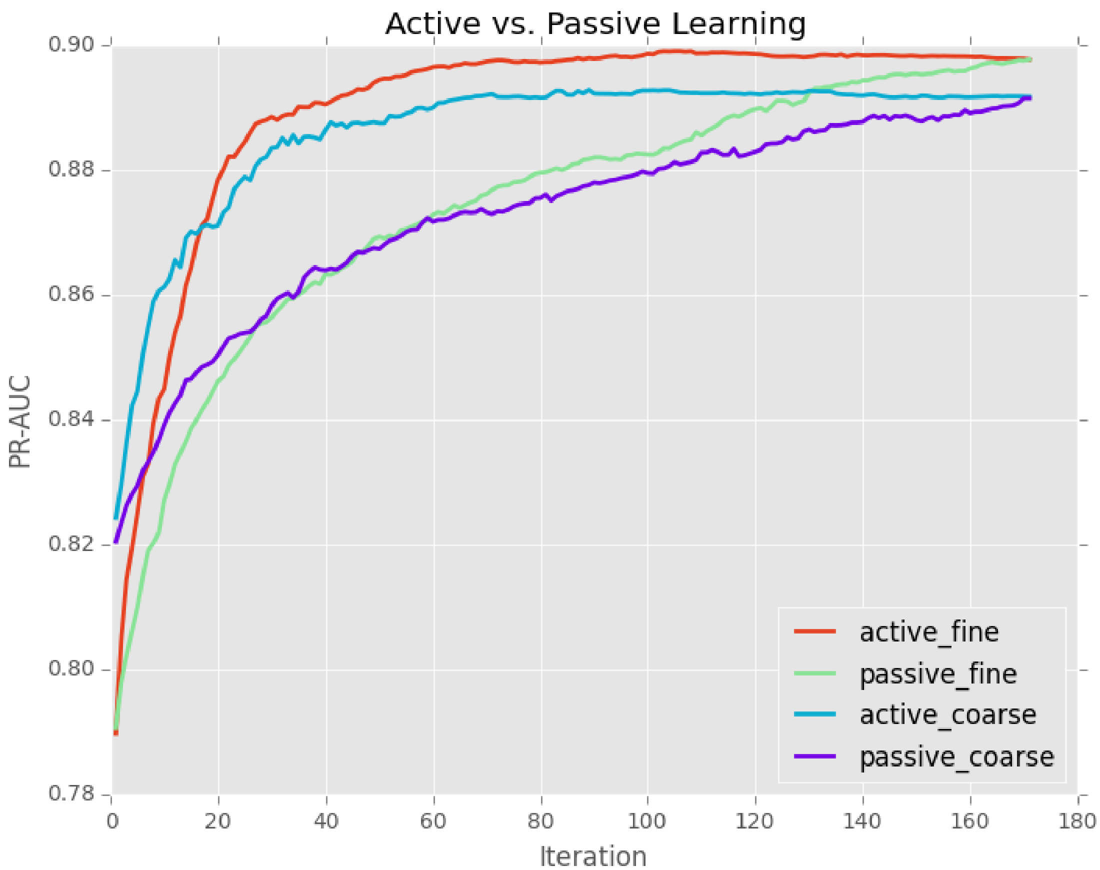


Figure 4.7: The PR AUC curves for rounds with SVM show little advantage for fine. The results are slightly different than the ones shown on 2/14 due to fixing a bug with the code that wasn't performing the preprocessing scaling for the SVM case at the same stage as it was being done for the logistic regression classifier.

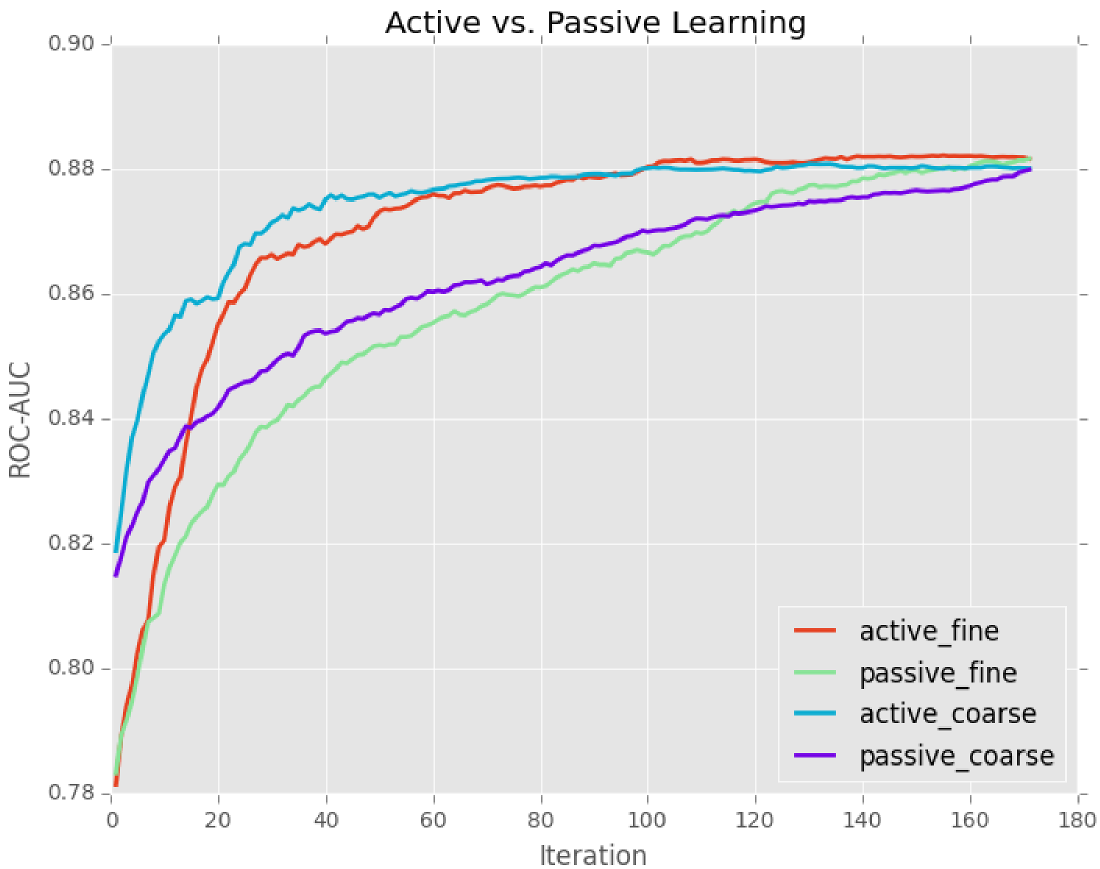


Figure 4.8: The ROC curves show more of an advantage for coarse classifiers.

4.3 Plots for FFR experiments

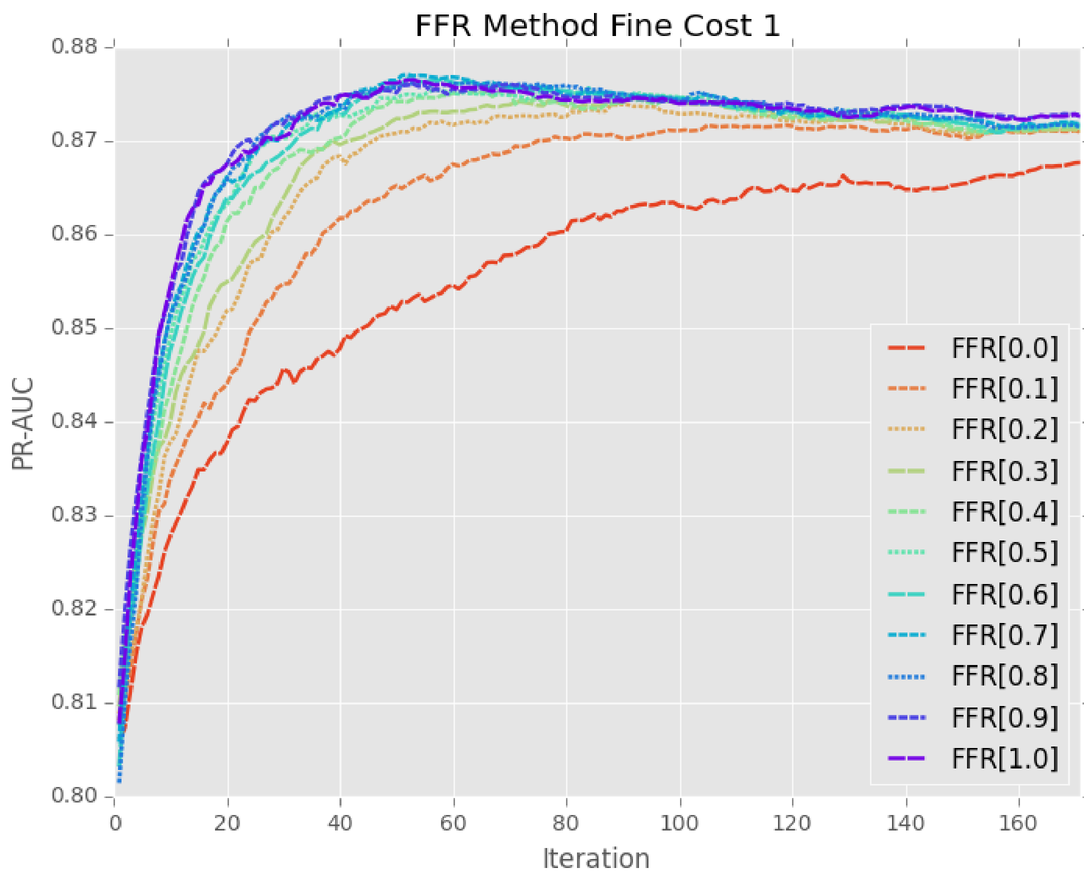


Figure 4.9: The strategy is changed from purchasing a set number of instances per round to having a set budget per round and spending a portion of that budget on fine and coarse grained labels. For this curve the fine and coarse grain labels both have a cost of 1. The purple 1.0 curve shows that if only fine grained labels are purchased, the highest performing PR-AUC can be obtained. The results are an average of 10 folds.

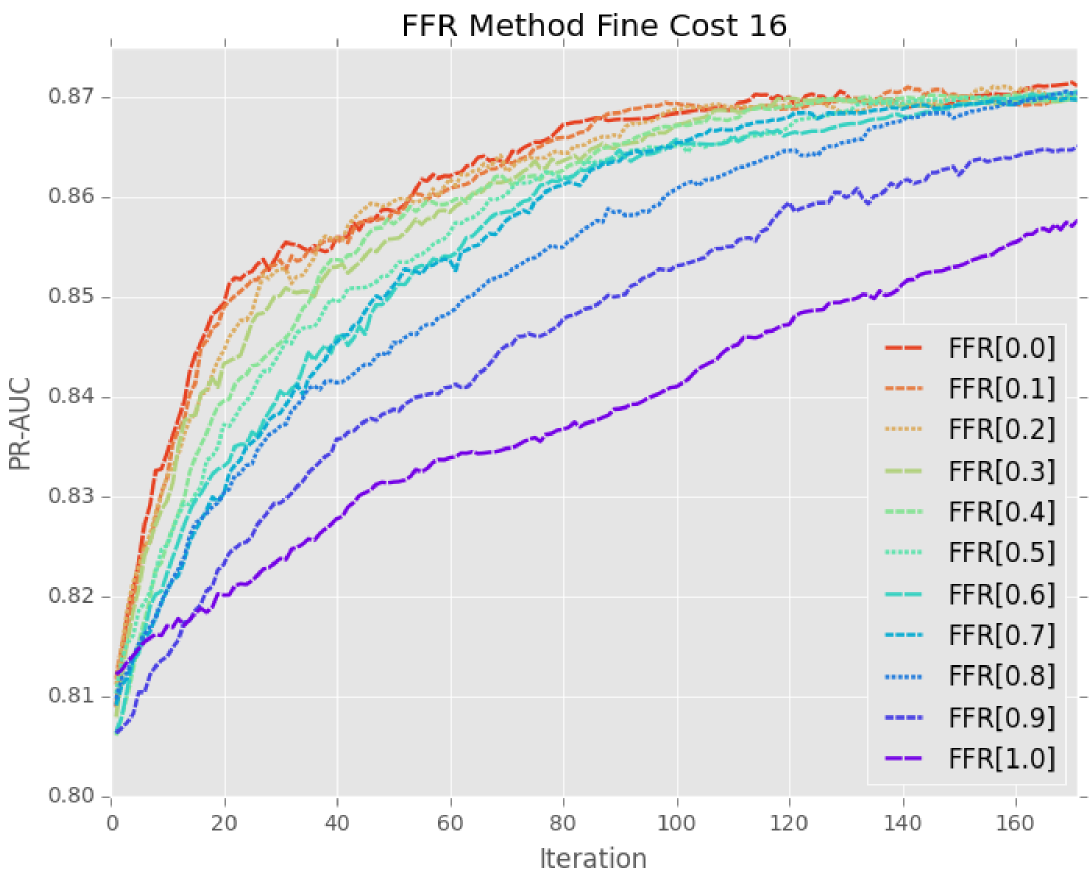


Figure 4.10: The fine cost is increased to 16. The budget for each iteration is 100, and for the case of the 0.5 curve, 50 instances are bought for coarse and 3.125 instances are bought for fine. The remainder 0.125 is then turned into a 0.125 chance for any round to purchase an extra fine label. The round size for the FFR 1.0 curve is very small, with only 7 labels purchased per iteration. The cost is to high for the fine label advantage to offset the decreased number of instances purchased.

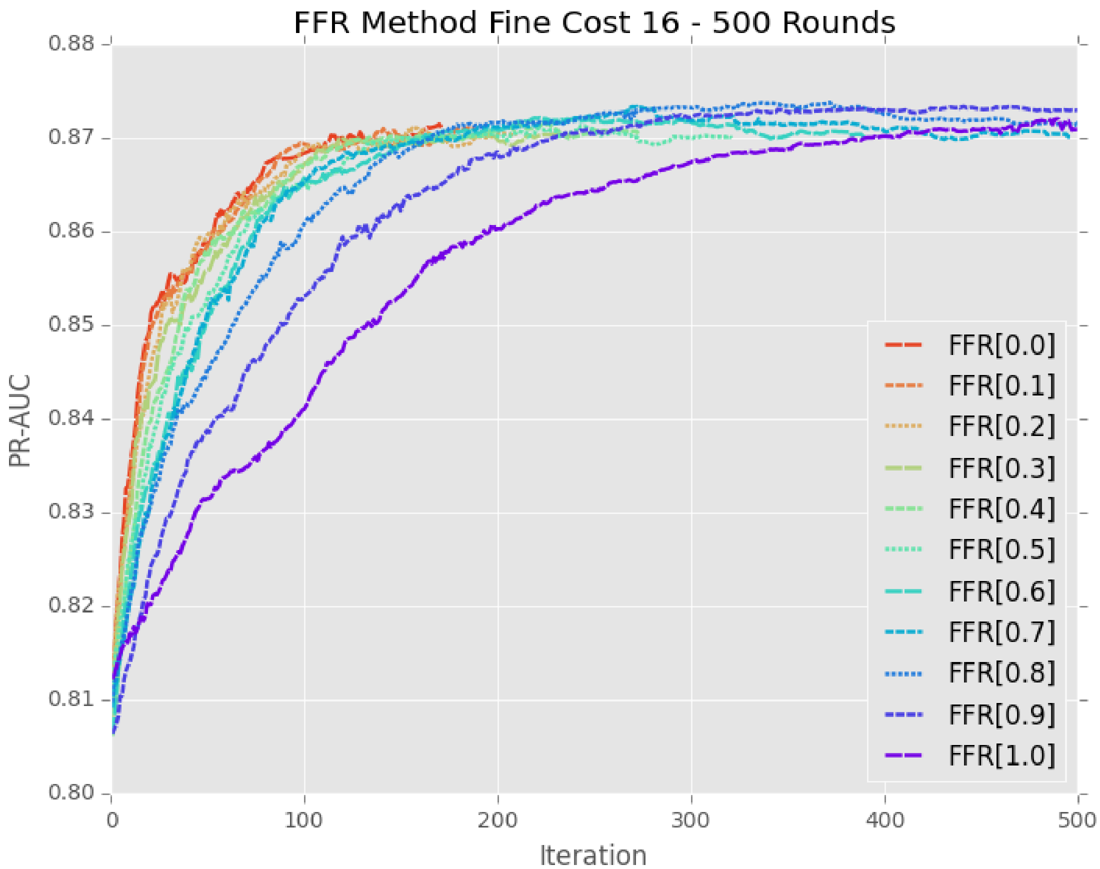


Figure 4.11: This shows the iterations continuing through round 500, the curves with the higher fine rates eventually settle to the same end point that the curves with the high rates of coarse labels purchased achieved at previous iterations.

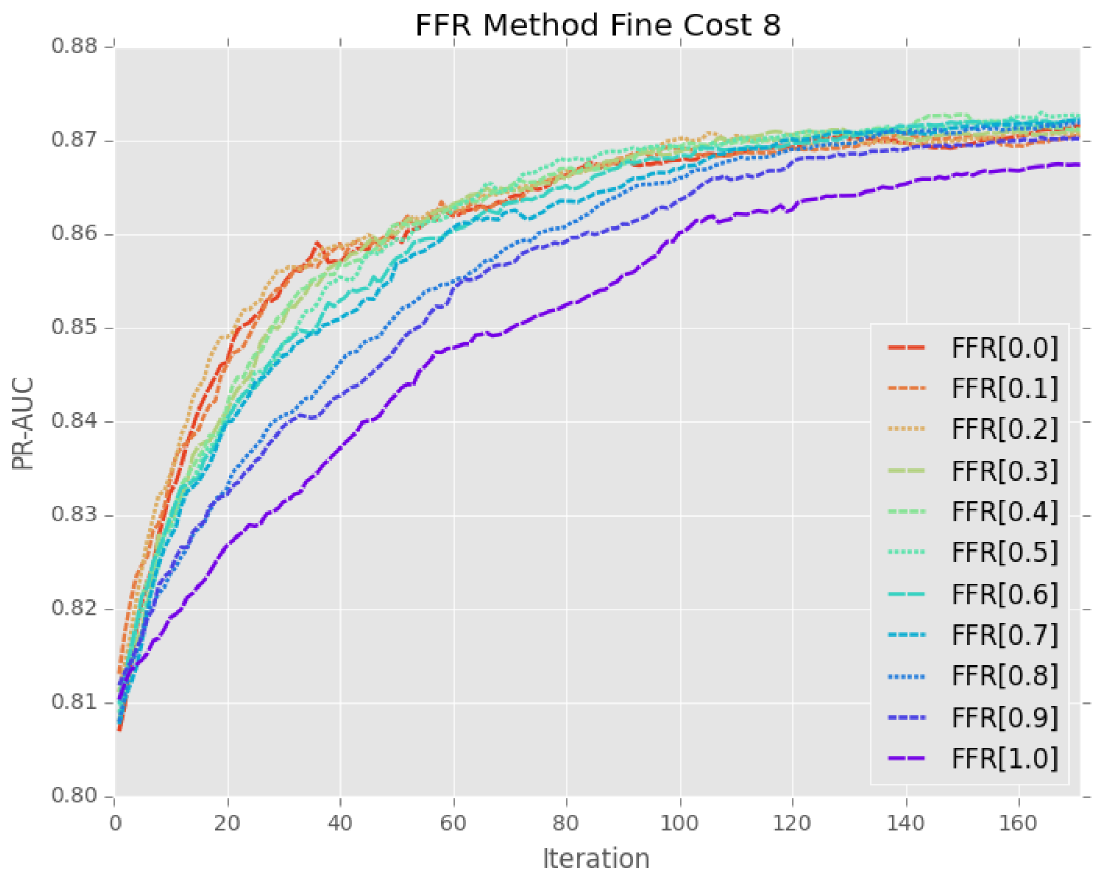


Figure 4.12: At fine cost 8 the FFR 0.0 rate is no longer the best option, 0.1 generally outperforms 0.0 slightly.

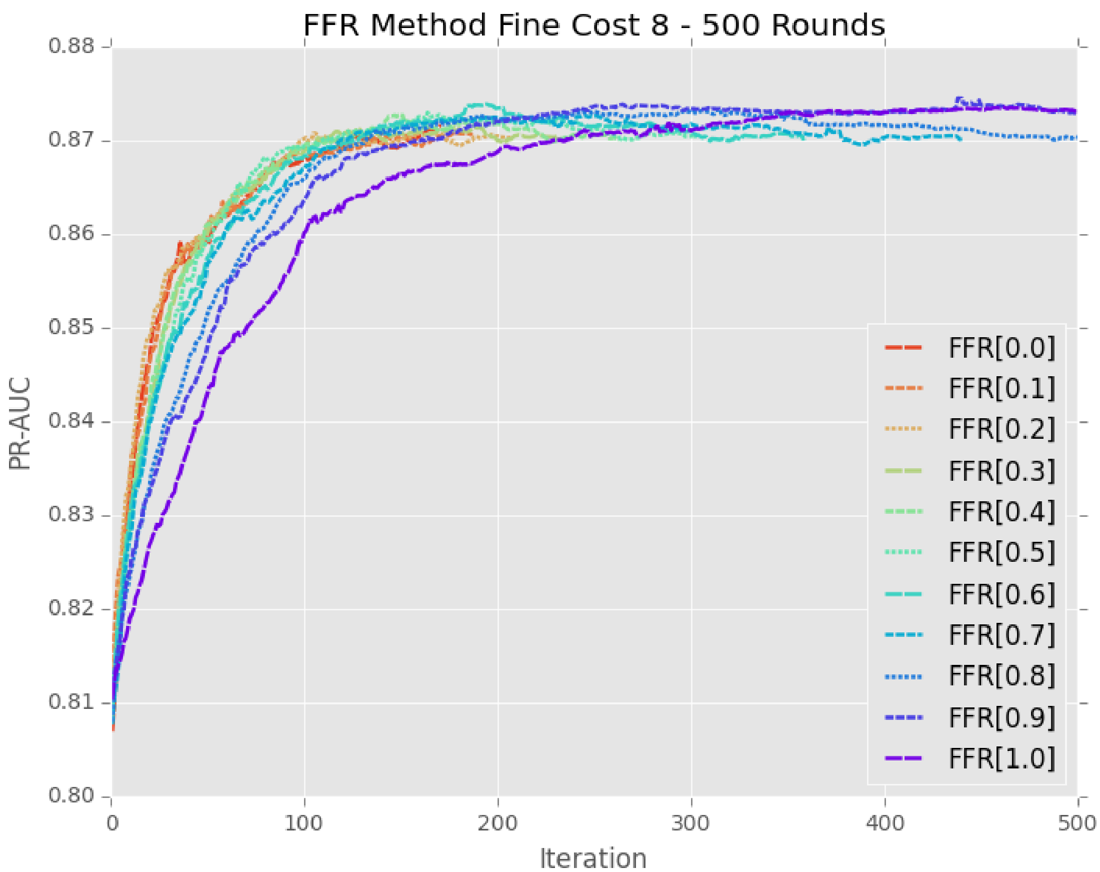


Figure 4.13: The extended picture of the FFR cost 8. The round size for FFR 1.0 is small, only 13 instances purchase per iteration.

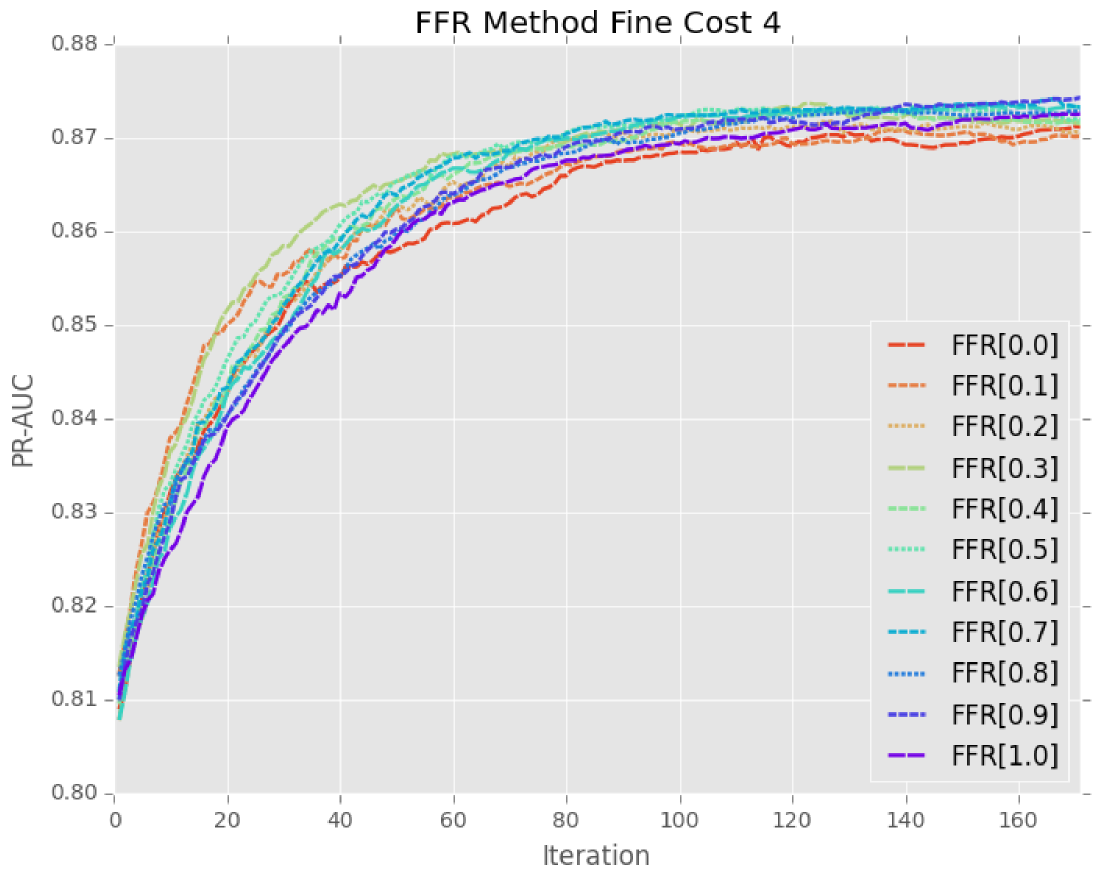


Figure 4.14: At fine cost 4, FFR 0.3 appears to be the highest performing rate.

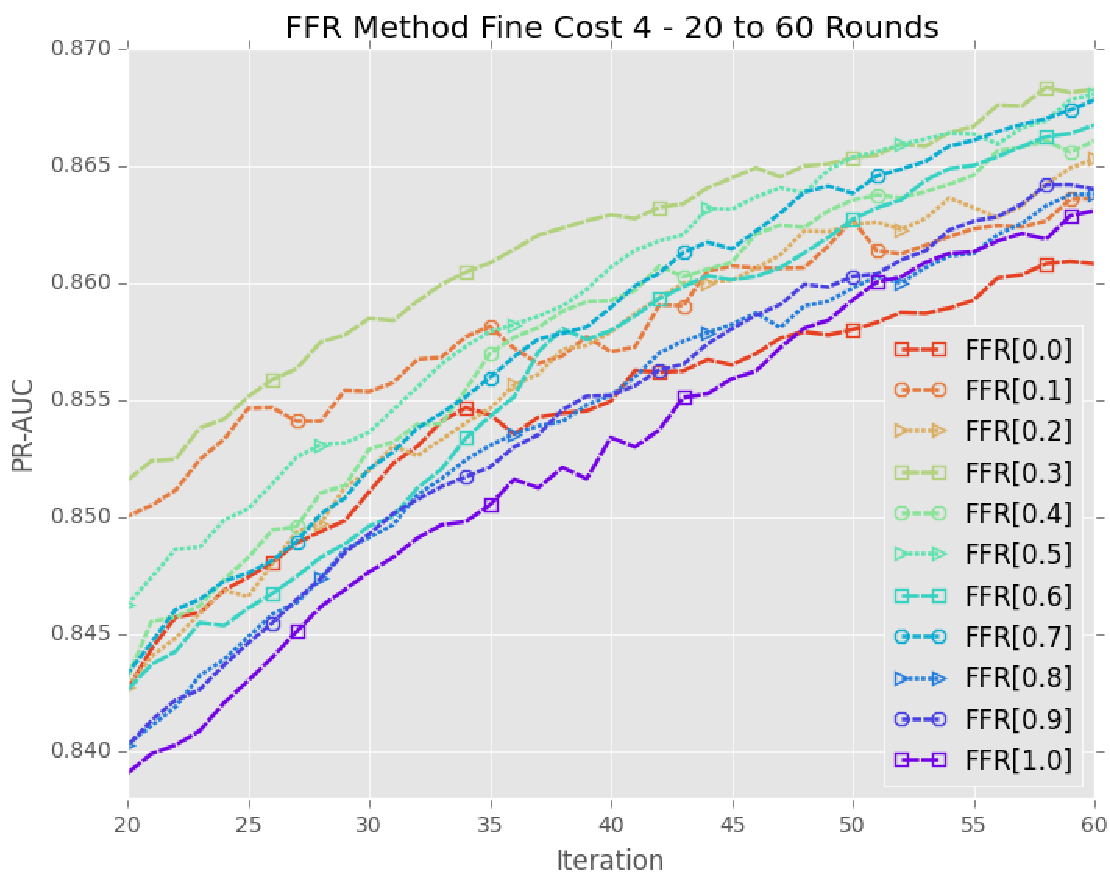


Figure 4.15: The fine cost 4 curves shown expanding the rounds 20-60.

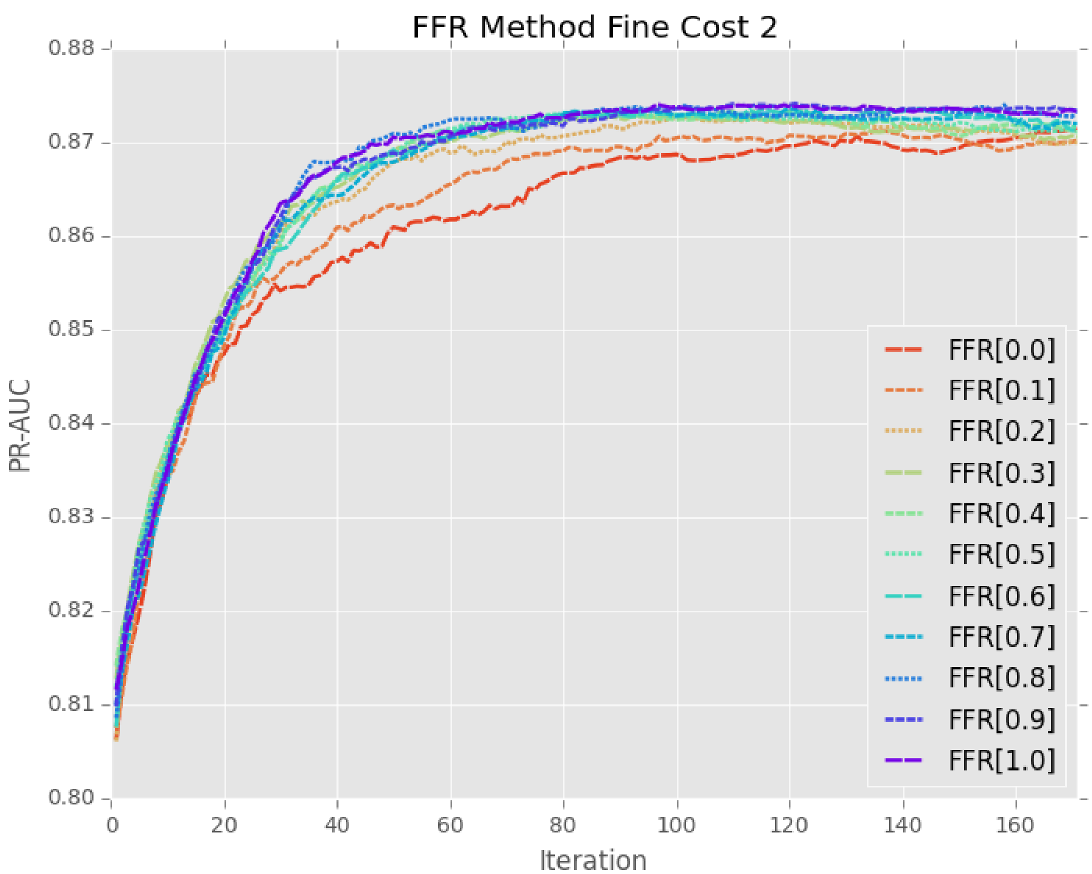


Figure 4.16: At fine cost 2, the preferred rate jumps up to 0.8, similar to the cost 1 results.

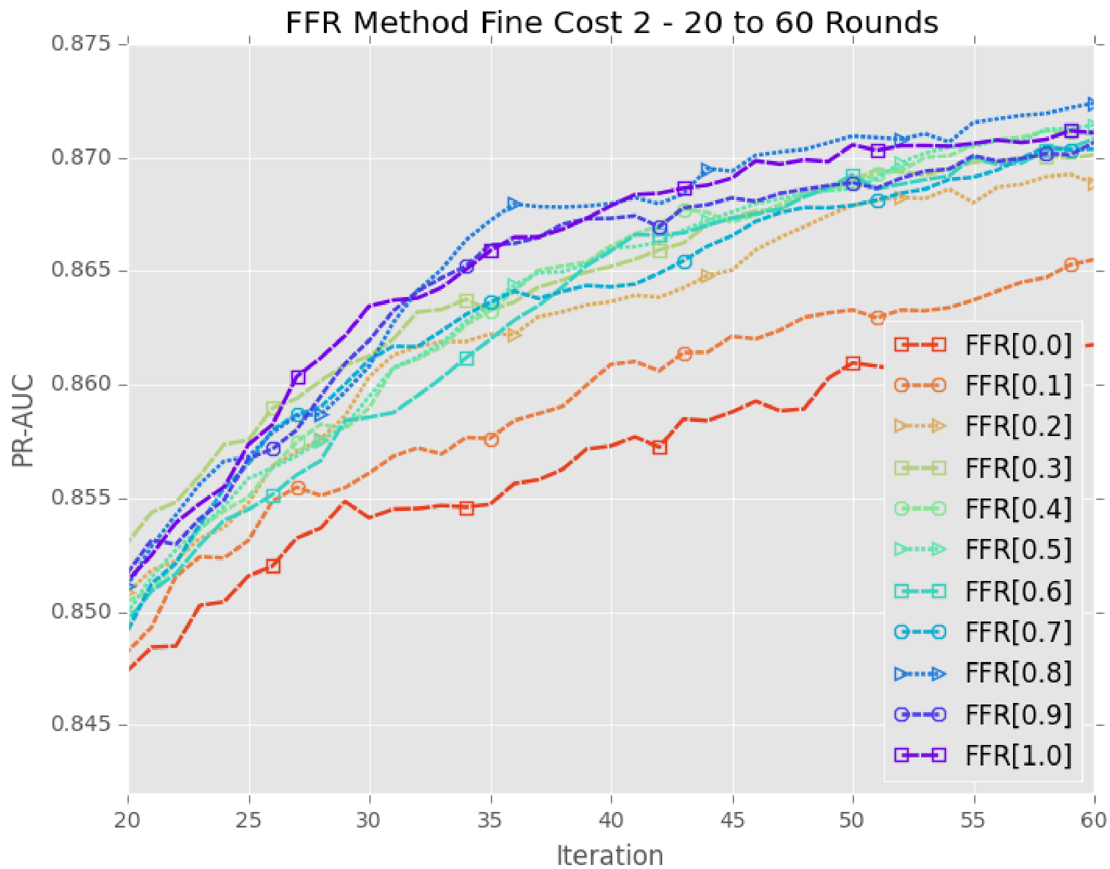


Figure 4.17: The fine cost 2 curves shown expanding rounds 20-60.

Chapter 5

Conclusions and Future Work

I should probably do the Bandit experiments.

Bibliography

[1] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997. 1.1

[2] 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, 3.1, 3.1.1

[3] W. Luo, A. Schwing, and R. Urtasun, "Latent structured active learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[4] 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

[5] 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

[6] X. Ling and D. Weld, "Fine-grained entity recognition," *Proceedings of the 26th Conference on Artificial Inteligence*, 2012. [Online]. Available: http://www.cs.washington.edu/ai/pubs/ling-aaai12.pdfhttp://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/download/5152/5124

[7] 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.