Dementia

March 15, 2021

Contents

1 Data Sources

1.1 Datasets Used

The datasets used are from the following dementia surveys conducted in Lebanon:

- Bekaa Questionnaire: This dataset is the most recent survey conducted last year in the Bekaa. This dataset involved 219 elderly.

- full validation data: This data consists of 281 rows. This is the very first survey that we implemented for the sake of testing the 1066 Dementia SPSS algorithm. This survey was conducted on a 1-1 ratio with clinically diagnosed demented patients.

- dementia data 502 baseline updated: This is the second survey implemented which involved 502 elderly people. The selection of elderly to participate in this survey was random.

We then merged this dataset into one dataset consisting of a total of 1024 elderly.

1.2 Dementia Output

In order to label each row (elderly) as positive (has dementia) or negative (does not have dementia), we run the 1066 Dementia SPSS algorithm on our dataset. The result was a total of 1024 patients, 203 of whom have dementia according to the SPSS algorithm.

2 Split Into numeric and textual

We have split the data into two chunks, one containing numeric features and another containing categorical features. The Bekaa and full validation questionnaires involved textual questions as well. However, these textual questions are not used by the dementia 1066 SPSS algorithm in determining whether the elderly has dementia or not, so we dropped these textual features from the analysis.

3 Numeric Features

Our data is now a mix of numerical and categorical features. For the numeric features, we have obtained the following information:

- **data type:** Type of the numeric feature. Could possible be:

  1. **numeric** meaning this feature has continuous range of values
  2. **categorical** meaning this feature has only a finite set of values, each resembling a specific category
  3. **ordinal** meaning, this feature's values are categorical, but they can be compared between each other. Example would be: a job, were a specific job might be values more than the other. Example: manager vs assistant

  the specification of the **data type** of each feature was done manually by looking at the range of values in each, and also by looking at the choices workbook in the copy of dementia Excel questionnaire

- **Description:** Description of each feature

- **cat_options:** Short for "categorical options". *In case the feature is categorical*, what are the possible categories.

- **val_range:** Short for "values range". *In case the feature is numerical*, what are the range of values this feature takes
- **min:** *In case the feature is numeric*, what is the minimum value it has
- **max:** *In case the feature is numeric*, what is the maximum value it has
- **distribution:** link to the distribution plot of each feature on bitbucket. The distribution plot helps in detecting erroneous data
- **perc_missing:** Short for "percentage of missing values". This helps us ggregate features with high percentage of missing. In case the percentage of missing is very high, Imputation methods won't aid much
- **perc_erroneous:** Short for "percentage of erroneous values". Detecting erroneous values was done manually for each feature. Anomalies cover uni-intentional or intentional erroneous values. Example would be having "age" feature being 1966 instead of the actual age
- **erroneous:** What were the erroneous values, if any
- **cut_off:** The cut off used to decide whether a feature's value is erroneous or not. The cut off was discovered manually by us for each feature.
- **color code:** a color code was applied to each feature to determine whether it belongs to the informant or not. The features having the color code yellow are **informant**, the ones having the color code red are related to the **patient** himself/herself. [1]

3.1 Informant and High Percentage of Missing

Most of the features associated with very high percentage of missing were directed to informants and related to the living conditions of the family, the house, income, cor-residents of the elderly, etc... which are not used by the SPSS algorithm to label the patients as demented or not (since they are not about the patients) **For this reason, we dropped the questions that were about the household, informants, and co-residents**

* informant is usually the caregiver of the interviewed elderly person

3.2 Nested Questions and High percentage of missing

Since our data is survey in nature, some features are actually questions are related to the other. If a certain question is asked to the elderly, and only based upon the elderly's answer, the interviewer might ask the elderly another question or not. Let us give an example:

[1] The dataset containing information about all numeric features can be found on our public repository on github: `https://github.com/hiyamgh/dementia/blob/master/input/codebooks/numeric.xlsx`

1. the interviewer asks if there has ever been a period when the elderly smoked cigarettes, cigars, a pipe, or water pipe nearly every day?

2. **Only if the answer to the question above was a Yes**

3. the interviewer asks the following questions:

   (a) What did the elderly smoke?

   (b) How old were you when you started using tobacco regularly? (if the above is true)

   (c) Do you still use tobacco regularly?

We also realized that many features have high percentage of missing because a parent question's answer did not cause a jump to that feature question.

3.3 Treating Features with High Percentage of Missing

We divided the features as follows:

1. **Informant** A feature belongs to this category if the question is asked to the informant

2. From Informants we have:

   (a) **parent** Questions that caused jumps to other questions

   (b) **child** child of parent question

3. **Non-Informant** We also have:

   (a) **parent**

   (b) **child**

We followed the following steps in order to eliminate features with high percentage of missing:

1. remove all informants and their children questions (so we removed all informants, and if one of them happens to be a parent question, we removed all its children questions)

2. It is important to note that all parent questions informants have their children questions also informant

3. Then we are left with **Legal features**

4. From legal features, remove all children

After doing the steps above, the number of numerical features decreased from 542 to 257

3.4 Detecting Erroneous Values Inside Features

In our dataset, the categorical and ordinal values are encoded with numbers ranging from 1 to the length of the categories. therefore, since our categorical, ordinal, and numerical values are all numbers, we set cut-offs to determine erroneous values. For some of the features, the erroneous values are straight forward to detect, based on the description for the question. These are the following:

- **age**: some people enter the year, others enter the age. for this, if the value of both the numb as is or 2020 - the numb is more than 100, this means that the person is more than 100 years old which renders the age erroneous. For age, the cut-off was 100
- **helphour**: number of hours per week the elderly needed help, therefore bounded by max numb of hours in a week
- **learn questions**: the patient is supposed to repeat 3 words, bounded by the max numb of words: 3

For the rest of the categorical and ordinal values, we get the maximum number encoding for the question options from the options excel sheet based on which the interviewers selected these options. The cut-off is therefore the maximum number encoding a category. Were therefore label any value that is greater than the cut-off to be erroneous.

3.5 Working with Legal Features

As mentioned in section 3.3, the remaining ***"Legal"*** Features are the 257 out of originally 542. We will be using these 257 ***"Legal"*** features as input for data pre-processing and cleaning techniques.

3.6 Filtering Legal Features with Erroneous Values - Erroneous Code-Book

In section 3, we talked about how some of the 542 numeric features have erroneous values. However, as we will be working with only "Legal" features, its important to filter out which subset of the features with erroneous values are actually "Legal". Thats's why, we did the following:

1. From all the features containing erroneous, filter out the ones that are Legal
2. From the Legal ones:
   (a) get the feature name
   (b) get the feature's description
   (c) get the feature's percentage of erroneous values

    (d) get the feature's actual erroneous values
    (e) Display the feature's cut-off on values which decides which values are erroneous (crossing the cut-off) and which are non-erroneous (not crossing the cut-off)

3. When done from 2. above, sort the features by decreasing order of percentage of erroneous values

3.7 Detecting Outliers

We want to investifgate the existsence of outliers in the data. We have 3 different data types:

1. **Numeric**: very few columns
2. **Ordinal**: categorical values that obey a certain order of importance
3. **Categorical**

The outlier detection methods are known for numeric values, but it is **ordianl** and **categorical** values that raise the question of how are we going to detect outliers with such types of data ?
We will assume that ordinal values are numeric. We will prove the legit-ability of our assumption through an example:
Assume for instance, that we have a column that shows the socio economic status ("low income","middle income","high income"), education level ("high school","BS","MS","PhD"), income level ("less than 50K", "50K-100K", "over 100K"), satisfaction rating ("extremely dislike", "dislike", "neutral", "like", "extremely like"). These are not categorical but rather ordinal, and if we give, for example, the income level the following labels:

- less than 50K: 0
- 50K-100K: 1
- over 100K: 2

Then it is definitely the case that $2 > 1 > 0$. And if we have an individual who earns, > 1000 K, we give this the label 3, and these individuals are definitely rare and can be considered as outliers

3.8 Outlier Detection and Scaling

We try different scaling methods before we detect outliers, and we apply the outlier detection only for columns *that are: numeric and/or ordinal*, and we extracted the percentage of outliers for each of the legal columns as well as the outlier values themselves (values considered outliers).
We realize that the outlier values, for the **ordinal** columns, happen to be the values *8 and 9* which is normal because the values are either 0,1,2 or 8,9.

3.9 Erroneous Codebook

All the work done in Section 3 is summarized in the a codebook which we called: erroneous codebook. [2]

The following is a description of each column in the codebook:

1. **COLUMN:** Name of the feature
2. **data_type:** type of the feature (specified manually by our team) numeric, ordinal, or categorical
3. **theme:** The survey theme of the feature. Extracted from the Copy of dementia baseline questionnaire. [3]
4. **description:** Description of the feature
5. **frequencies**: the unique values in the feature and their frequency (in percentage)
6. **perc_missing:** percentage of missing values in the feature
7. **erroneous**: the erroneous values in the feature, if any.
8. **perc_errnoneous:** percentage of erronous values, if any
9. **cut_off:** cut off value based on which we selected the erroneous values in a feature. This was done manually by our team (refer to Section 3.4 for more details)
10. **outliers_zscore:** percentage of outliers identified using z-score
11. **outliers_iqr:** percentage of outliers identified using IQR

[2]The erroneous codebook can be found here on our public github repository: `https://github.com/hiyamgh/dementia/blob/master/input/codebooks/erroneous_codebook_legal_outliers_filtered.csv`

[3]Copy of dementia survey can be found here on our public github repository: `https://github.com/hiyamgh/dementia/blob/master/input/~%24Copy%20of%20Dementia_baseline_questionnaire_V1.xlsx`

3.10 Data Preprocessing

Our legal features that we have filtered contain a mix of numeric, categorical, and ordinal features. By referring to the erroneous codebook [2], we have created all the features have missing values, we have done the following:

3.10.1 Imputing Missing Values

- **Numeric & Ordinal** We have imputed missing values using KNN Imputation
- **Categorical:** We have imputed missing values by creating a new category for the missing features. This decision is based on the fact that we cannot impute categorical features using KNN due to the discrete nature of categorical values.

3.10.2 Scaling Features

Due to the discrete nature of values in the categorical features, **we scaled only the numeric and ordinal features**. We have used Robust Scaling from python's *sklearn* module. Robust Scaling is well suited for features with outlier and we do have outliers.

4 Feature Selection

4.1 Decision Tree Feature Importance

We use decision tree regressor to calculate feature importance implemented using scikit-learn's DecisionTreeRegressor.

After being fit, the model provides a feature_importances_ property that can be accessed to retrieve the relative importance scores for each input feature.

The results of the feature importance using the dummy imputed pooled data so far is then stored in the 'feature_importance.csv', sorted from highest (highest importance score) to lowest (lowest importance score).

4.2 Select K Best Features

4.2.1 Deciding "K"

In order to select K best features, we need to first decide the value of K. In order to do this, we use sklearn's RFECV, which uses step forward recursive feature elimination (1-step) and cross validation to decide the number of features needed to give the highest out of sample accuracy. The model used for this was decision tree regressor.

4.2.2 Selecting K Best Features

After deciding on the value of K, which using the current imputation of the input data turned out to be 20, we perform step forward recursive feature elimination that ranks all the features as 0 or 1, with 1 meaning that a feature is one of the K best features. We store the names of the k-selected features (column names) into a csv file called "k_best_features.csv" This is also performed using a decision tree regressor.

5 Oversampling and Undersampling

Based on the examples discussed in the books, we implemented 4 functions that accept a model and add it to a pipeline after adding the oversampling/undersampling layers. The following are the functions created:

1. random_sampling: consists of a random oversampling and a random undersampling layers *works with mixed data types*
2. smote_random_sampling: adds a smote oversampler and a random undersampler *works with mixed data types*
3. smote_tomek (combination approach): combines over and under sampling using SMOTE and Tomek links. *could not find support for mixed datatypes*
4. smote_enn: combines over and under sampling using SMOTE and Edited Nearest Neighbours. *could not find support for mixed datatypes*

5.1 Dealing With Mixed Variables

The python library that is used in the book provides a model, SMOTENC, that handles mixed datatypes. For this model, the only difference is that we have to specify the indices of the categorical variables. The random sampler model can also be used for mixed datatypes. SMOTENC does not require scaling but according to my search it is adviced to scale the numerical variables using either MinMax or standard deviation scaling, and let the SMOTENC handle the categorical variables.

6 Cost-Sensitive Learning

6.1 Cost sensitive learning consists of the following steps:

1. Input is the imputed and scaled data.
2. Built a pipeline that consists of SMOTENC and the model of choice.
3. Pipeline is passed into a GridSearchCV, to find the best percentage of sampling for the SMOTE, and the best params for the model of choice

using GridSearch and cross validation. The scoring metric for finding the best combination of params was AUC ROC.

4. cross validation was achieved through RepeatedStratifiedKFold

5. The above steps are repeated for every tested model, followed by reporting the results of the test dataset.

6. The results (metrics) are: ROC AUC, GMEANS, f2-score, BSS, AUC PR, cost matrix

7. BSS: is calculated by using a reference score, and comparing the score of the model to that reference score (baseline score). BSS = 1 - (model BrierScore/ reference BrierScore). If the reference score was evaluated, it would result in a BSS of 0.0. This represents a no skill prediction. Values below this will be negative and represent worse than no skill. Values above 0.0 represent skillful predictions with a perfect prediction value of 1.0. The book suggests the reference score to be the brier score of a probability of 0.01 for all rows (not an actual model predicting the probabilities)

8. The tested models are: XGBoost, KNeighbors Classifier, Balanced Random Forest (from the imbalanced library), Weighted Logistic Regression, Weighted Decision Tree,Weighted SVM

6.2 Tested sampling methods (using SMOTENC's sampling_strategy options):

1. minority: resample the minority class (oversampling)

2. not minority: resample the majority class (undersampling)

3. all: resample both classes

4. When float, it corresponds to the desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling, this ratio is reached by oversampling the minority class to reach the desired ratio. Tested ratios: 1, 0.5, 0.75

6.3 Tested class_weights: (costs for each label), combination of suggestions from book and built-in options:

In the below list, cost(1) means cost of false positive and cost(0) means cost of false negative.

1. reverse: cost(1)= number of 0s in the train and cost(0) = number of 1s in the train. This is suggested by the book: invert the ratio of positive-to-negative and used as the cost of misclassification errors, where the cost of a False Negative is the length of negative in the dataset and the cost of a False Positive is the length of the negative in the original dataset.

the book suggests that it is a good idea to use this heuristic as a starting point, then test a range of similar related costs or ratios to confirm it is sensible.

2. cost(1)= 10, cost(0)=1
3. cost(1) = 100, cost(0)=1
4. cost(0)= 10, cost(1)=1
5. cost(0) = 100, cost(1)=1
6. balanced: The "balanced" mode provided by the library uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))

6.4 Probabilistic Models

6.4.1 Grid Search For Optimal Probability Threshold

As suggested by the book, after performing grid search, training, and testing for each model, we test a range of probability thresholds to find the one that gives the highest f2 score. The range is from 0 to 1 with 0.001 steps.

6.4.2 Calibrated Models

This is provided by SKLEARN library, we specify class weights for each class in the label which allows for cost sensitive learning (as done in the book). Calibration was using sklearn's CalibratedClassifierCV, which takes a model and performs its regular grid search to find the best parameters but also uses cross validation to calibrate the model.

6.4.3 Balanced Random Forest Results

The Balanced Random Forest class from the imbalanced-learn library performs data sampling on the bootstrap sample in order to explicitly change the class distribution and performs random undersampling of the majority class in each bootstrap sample. This is generally referred to as Balanced Random Forest. It takes care of the class_weight (cost matrix) and does not need it explicitly specified since it expects imbalanced datasets, Unlike other models provided by sklearn that need the cost matrix for imbalanced classification.

6.4.4 Results

Model Name	f2-score	G-MEAN	BSS	PR AUC	ROC AUC	sampling strategy	cost matrix
baseline logistic Reg.	0.66	0.79	0.55	0.56	0.79	-	-
Calibrated Weighted Logistic Reg.	0.752	0.808	0.273	0.427	0.810	both	reverse
Calibrated Weighted Decision Tree	0.722	0.747	0.321	0.350	0.758	over-sampling	reverse
Balanced Random Forest	0.768	0.797	0.418	0.398	0.805	under-sampling	-

6.4.5 Results With Top 10 Features

Model Name	f2-score	G-MEAN	BSS	PR AUC	ROC AUC	sampling strategy	cost matrix
baseline logistic Reg.	0.657	0.786	0.548	0.560	0.792	-	-
Calibrated Weighted Logistic Reg.	0.728	0.821	0.300	0.528	0.821	ratio=0.75	reverse
Calibrated Weighted Decision Tree	0.647	0.756	0.353	0.402	0.756	both	reverse
Balanced Random Forest	0.727	0.815	0.379	0.491	0.815	under-sampling	-

6.4.6 Results With Top 20 Features

Model Name	f2-score	G-MEAN	BSS	PR AUC	ROC AUC	sampling strategy	cost matrix
baseline logistic Reg.	0.657	0.786	0.548	0.560	0.792	-	-
Calibrated Weighted Logistic Reg.	0.758	0.814	0.293	0.437	0.816	over-sampling	reverse
Calibrated Weighted Decision Tree	0.759	0.801	0.390	0.409	0.806	ratio=1	reverse
Balanced Random Forest	0.822	0.876	0.457	0.567	0.876	under-sampling	-

6.5 Classfication Models

6.5.1 Results

Model Name	f2-score	ROC AUC	sampling strategy	cost matrix
KNeighbors	0.709	0.763	over-sampling	-
Calibrated Weighted SVM	0.746	0.787	ratio=0.75	reverse
Easy Ensemble Classifier	0.729	0.791	under-sampling	-

6.5.2 Results With Top 10 Features

Model Name	f2-score	ROC AUC	sampling strategy	cost matrix
KNeighbors	0.658	0.759	over-sampling	-
Calibrated Weighted SVM	0.725	0.825	ratio=0.75	cost(0)=1,cost(1)=10
Easy Ensemble Classifier	0.702	0.790	ratio=0.5	-

6.5.3 Results With Top 20 Features

Model Name	f2-score	ROC AUC	sampling strategy	cost matrix
KNeighbors	0.705	0.793	both	-
Calibrated Weighted SVM	0.738	0.801	over-sampling	cost(0)=1,cost(1)=10
Easy Ensemble Classifier	0.687	0.767	under-sampling	-

6.6 One-Class Classification

Model Name	f2-score
One class SVM	0.361
Elliptic Envelope	0.152
Isolation forest	0.261
Local Outlier Factor	0.269

7 SHAP

7.1 Global SHAP Summary Values Results

The summary plot combines feature importance with feature effects. Each point on the summary plot is a Shapley value for a feature and an instance. The position on the y-axis is determined by the feature and on the x-axis by the Shapley value

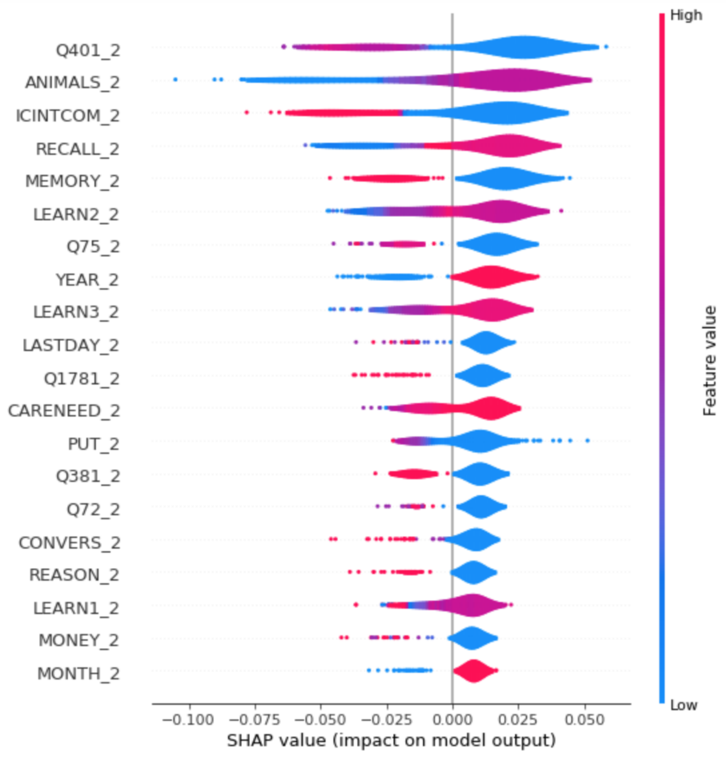


Figure 1: Class 0

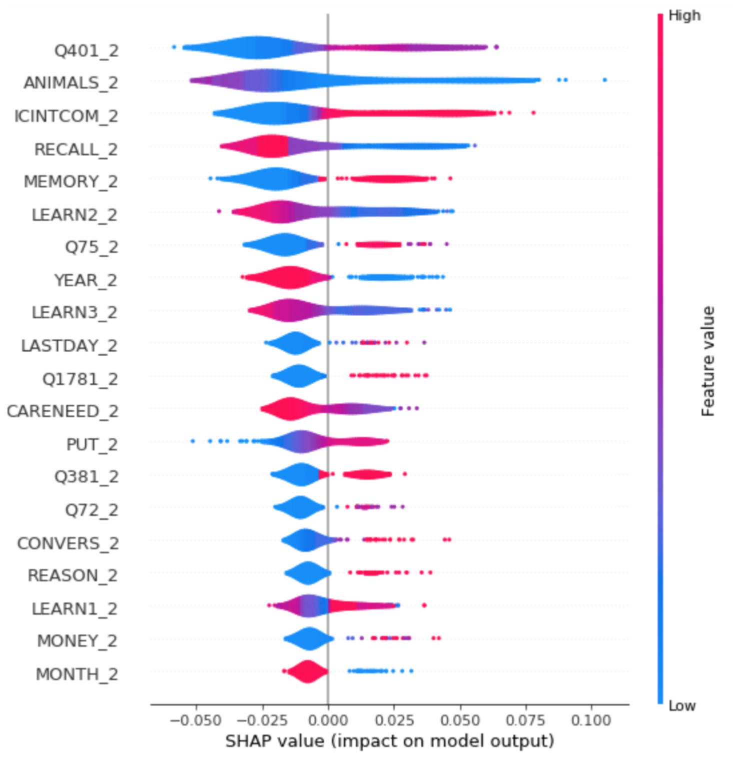
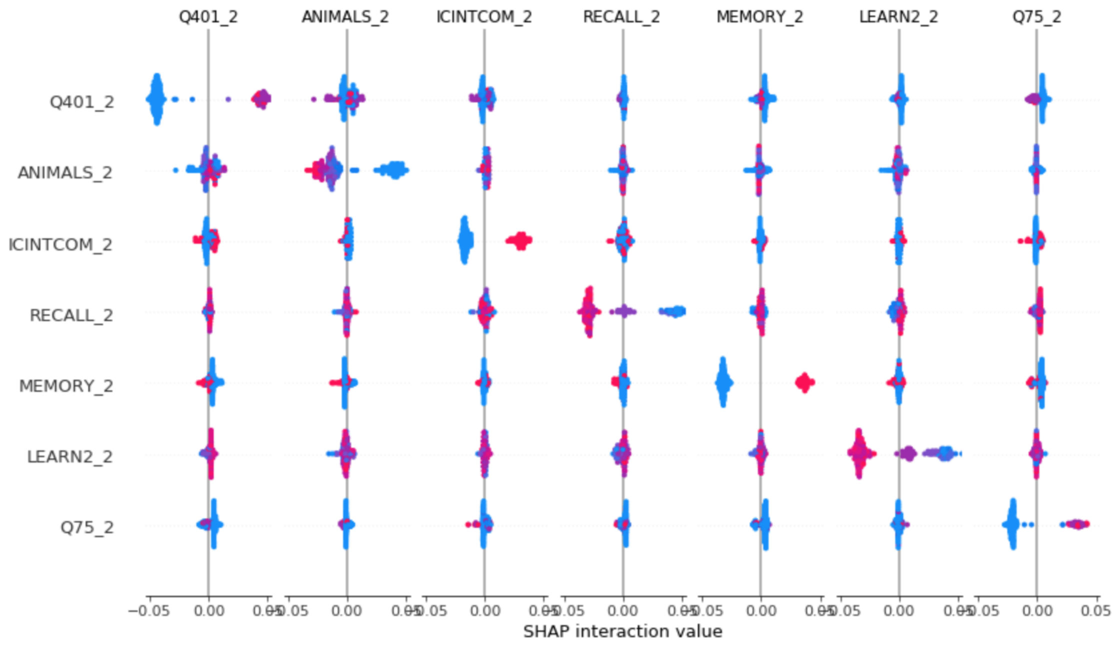


Figure 2: Class 1

7.2 Global SHAP Interaction Values



7.3 Multi-output Decision Plot for properly classified Rows

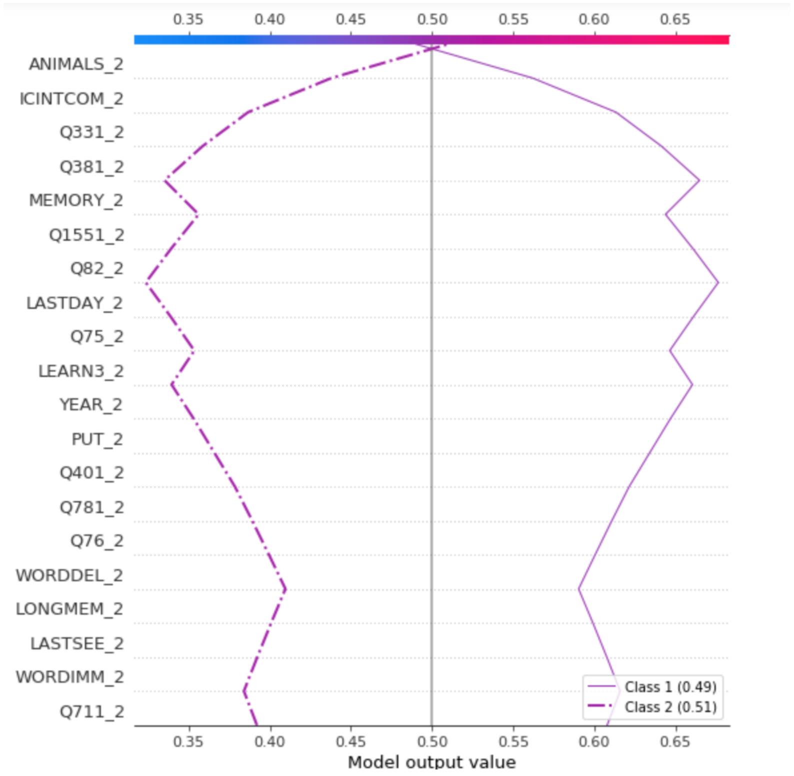


Figure 3: Row with label = 0

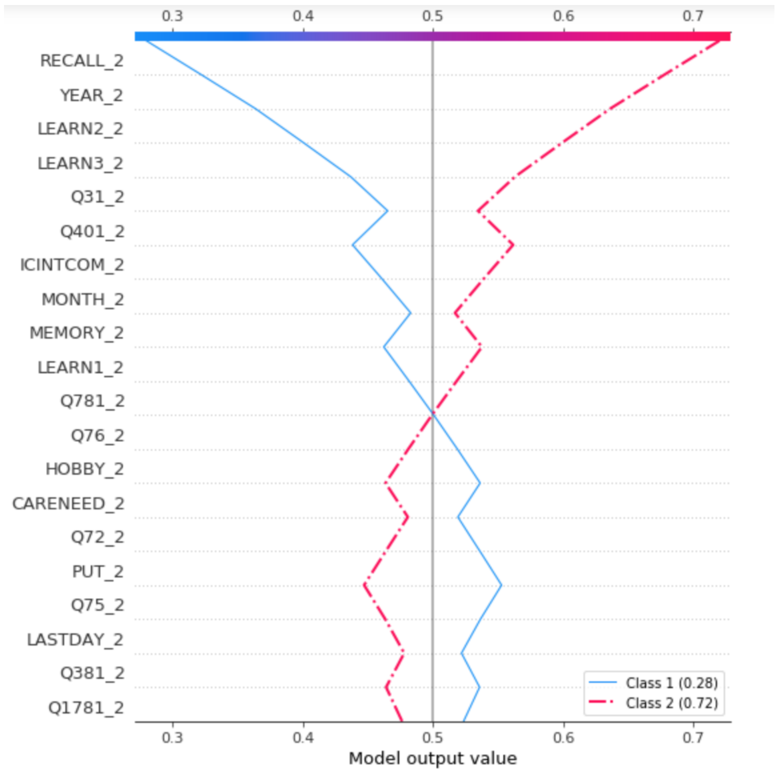
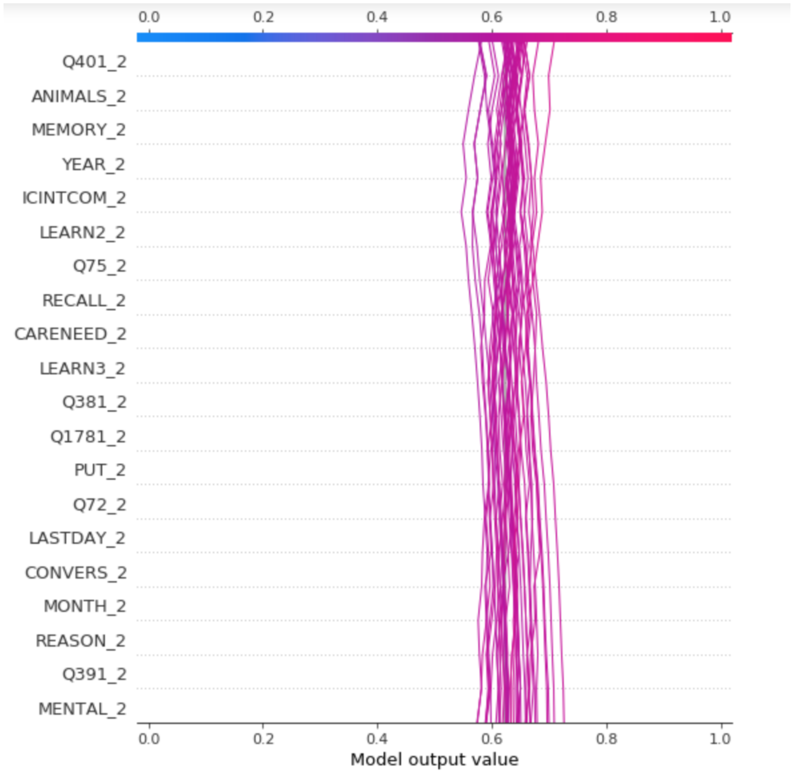


Figure 4: Row with label = 1

7.4 Decision Plot for Miss-classified Row



8 Chapter 6: Precision, Recall, and F-measure

The chapter summarizes the precision, recall, and f-measures of multiclass and binary classification for the case of balanced data.

9 Chapter 7:ROC Curves and Precision-Recall Curves

ROC Curves (receiver operating characteristic curve) and ROC AUC (area under the ROC curve) are used to decide if the model differentiates between the labels.

9.1 ROC Curve

The threshold is applied to the cut point in probability between the positive and negative classes. In order to decide on the best threshold, we evaluate the true positive and false positives for different threshold values, a curve can be constructed that stretches from the bottom left to top right and bows toward the top left. This curve is called the ROC curve. A classifier that has no discriminative power between positive and negative classes will form a diagonal line between (0,0) and (1,1). Models represented by points below this line have worse than no skill.

9.2 AUC

Area under the curve is calculated to give a single score for a classifier model across all threshold values. This is called the ROC area under curve or ROC AUC or sometimes ROCAUC. The score is a value between 0.0 and 1.0, with 1.0 indicating a perfect classifier.

10 Chapter 8: Probability Scoring Methods

10.1 Probability Metrics

On some problems, a crisp class label is not required, and instead a probability of class membership is preferred. The probability summarizes the likelihood (or uncertainty) of an example belonging to each class label.

10.2 LogLoss Score

Logarithmic loss or log loss for short is a loss function known for training the logistic regression classification algorithm. The log loss function calculates the negative log likelihood for probability predictions made by the binary classification model. Most notably, this is logistic regression,but this function can be used by other models, such as neural networks, and is known by other names, such as cross-entropy.

11 Cross Validation for Imbalanced Datasets

The solution is to not split the data randomly when using k-fold cross-validation or a train-test split. Specifically, we can split a dataset randomly, although in such a way that maintains the same class distribution in each subset. This is called stratification or stratified sampling and the target variable (y), the class, is used to control the sampling process. This is available using sklearn stratified kfold.

12 Chapter 10

Summarizes oversampling and undersampling techniques which are available in chapters 12 and 13 in details.

13 Chapter 12

13.1 Oversampling

All techniques available through the python library: imblearn.over_sampling

- Random Oversampling: simplest oversampling method, involves randomly duplicating examples from the minority class in the training dataset
- SMOTE (Synthetic Minority Oversampling Technique): works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample as a point along that line
- Borderline-SMOTE: involves selecting those instances of the minority class that are misclassified, such as with a k-nearest neighbor classification model, and only generating synthetic samples that are difficult to classify. Borderline Oversampling is an extension to SMOTE that fits an SVM to the dataset and uses the decision boundary as defined by the support vectors as the basis for generating synthetic examples, again based on the idea that the decision boundary is the area where more minority examples are required.
- Adaptive Synthetic Sampling (ADASYN): another extension to SMOTE that generates synthetic samples inversely proportional to the density of the examples in the minority class. It is designed to create synthetic examples in regions of the feature space where the density of minority examples is low, and fewer or none where the density is high.

13.2 Undersampling Techniques

- Random Undersampling

14 Data Transforms (Hiyam)

14.1 Scaling Numeric Data (chapter 17)

- Normalization (Min-Max Scaling)
- Standardization

14.2 Scaling Data with Outliers (chapter 18)

Many machine learning algorithms perform better when **numerical** input variables are scaled. Standardization is a popular scaling technique that substracts the mean from the values and divide by the stadard deviation, transforming the probability distribution from for an input variable to a Standard Gaussian (zero mean and unit variance).

Problem: Standardization can become skewed or biased when the input variable contains outliers.

Robust Scaling

When we are scaling the data, and if we have very large values relative to the other input variables, these large values can dominate or skew some machine learning algorithms. **The result is that the algorithms pay most of their attention on the large values and ignore the variables with smaller values**.
Outliers are values at the edge of the distribution that may have a low probability of occurence, yet are overrepresented for some reason. **Outliers can skew a probability distribution and make data scaling using standardization difficult as the calculated mean and standard deviation will be skewed by the presence of outliers.**
Approach: When standardizing input variables containing outliers, we can ignore outliers from the calculation of the mean and standard deviation, and use the calculated values to scale the data
This is called robust standardization. This can be achieved by calculating the median (50th percentile) and the 25th and 75th percentile. The values of each variable can then have their median subtracted and are divided by the inter quartile range (IQR) which is the difference between the 25th and 75th percentile.

$$value = \frac{value - median}{p_{75} - p_{25}} \tag{1}$$

The resulting value has a **zero mean and median and a standard deviation of 1**. Although not skewed by outliers and the outliers are still present with the same relative relationships to other values.

14.3 How to Encode Categorical Data (chapter 19)

Machine learning models require all input and output variables to be numeric. This means that if the data contains categorical data, we must encode it to numbers before we fit and evaluate our models.

Ordinal Encoding

Each unique category is assigned an integer value. Example: *red* is 1, *green* is 2, *blue* is 3.
For categorical variables, it imposes and **ordinal relationship** when no such relationship exists. This may cause problems and **one hot encoding** may be used instead.

One Hot Encoding

For categorical variable where no ordinal relationship exists, the ordinal encoding is not enough and may be misleading.
One Hot Encoding works by creating binary variables for each category. For Example: *red* will be [1, 0, 0], *blue* will be [0, 1, 0], and *green* will be [0, 0, 1].

Dummy Variable Encoding

The one hot encoding includes a binary representation for each category. This might cause redundancy.
if we know that [1, 0, 0] represents *blue*, and [0, 1, 0] represents *green*, we don't need another binary variable to represent *red*, instead we could use 0 values along, e.g. [0, 0]. This is called dummy variable encoding and always represents C categories with $C - 1$ binary variables.
Other being less redundant, it is required for some models.

14.4 How to Make Distributions Look More Gaussian (chapter 20)

Several Machine Learning Algorithms assume the numerical values have a Gaussian distribution. Our data may not have a Gaussian distribution and it might have a Gaussian-like distribution.

15 Data Pre-Pocessing

After we have filtered the legal features in our data, and we deleted the features that have > 40 % missing values, we are left out with 255 legal features. These features must be pre-processed before we do the modelling exercise, for the following reasons:

1. All the 255 legal features still have missing values
2. We have a combination of numeric, ordinal, and categorical features, whereby each might require a different kind of scaling

15.1 Imputing Missing Values - Categorical Features

For **all** categorical features, we will impute missing values by adding **a new category** for missing values. This way, any value that is missing will have its own category(label)

15.2 Imputing Missing Values - Numerical/Ordinal Features

For all numeric/ordinal features, we will impute missing values using **KNN**.

We discussed in one of our meeting that we will treat ordinal features as numeric, that's why we considered treating ordinal as numeric features therefore they are also imputed using KNN

15.3 Scaling - Numeric/Ordinal Features

We applied **Robust Scaling** for all numeric/ordinal features.
Robust Scaling is good for data that includes outliers, and all our features do have outliers (with a very low percentage, though).
For more information about the **percentage of outliers/feature, please consult with the following dataframe on our github repository:** `https://github.com/hiyamgh/dementia/blob/master/input/codebooks/erroneous_codebook_legal_outliers_filtered.csv`

15.4 Scaling - Categorical Features

No scaling is applied for categorical features.

16 Imputing Numerical Missing Data

In this section, we will be listing the multiple imputation methods that may be used for imputing missing **numeric** data. I referred to the data preparation book we have

16.1 Statistical Imputation

Using mean, median, mode, constant value

16.2 K Nearest Neighbors

K nearest neighbor model. A new sample is imputed by finding samples in the training set closest to it and averages these nearby points to fill in the value. Must specify the **distance** to use and the **number of neighbors** neeeded for imputation

16.3 Iterative Imputation

. Iterative imputation refers to a process where each feature is modeled as a function of the other features, e.g. a regression problem where missing values are predicted. Each feature is imputed sequentially, one after the other,allowing prior imputed values to be used as part of a model in predicting subsequent features.It is iterative because this process is repeated multiple times, allowing ever improved estimates of missing values to be calculated as missing values across all features are estimated. This approach may be generally referred to as fully conditional specification (FCS) or multivariate imputation by chained equations (MICE).

17 Imputing Categorical Missing Data

In this section, we will be listing the multiple imputation methods that may be used for imputing missing **categorical** data

Single based Imputation Methods

17.1 Mean Imputation

The mean imputation replaces missing values with the observed mean of the available data.

17.2 Imputation Using Most Frequent or (Zero/Constant Values)

Most Frequent is another statistical strategy to impute missing values and it works with categorical features (strings or numerical representations) by replacing missing data with the most frequent values within each column

17.3 Create a New Category (Random Category) for NAN Values

I think its more of a "hack" rather than an actual imputation method

17.4 Adding a Variable To Capture NAN

Replace NAN categories with most occurred values, and add a new feature to introduce some weight/importance to non-imputed and imputed observations. Create a new column and replace 1 if the category is NAN else 0. This column is an importance column to the imputed category.Replace NAN value with most occurred category in the actual column.

17.5 Imputation Using Deep Learning (Datawig)

Its actually a repository for AWS – DataWig learns Machine Learning models to impute missing values in tables. – So not sure how trustworthy it is This method works very well with categorical and non-numerical features. It is a library that learns Machine Learning models using Deep Neural Networks to impute missing values in a dataframe. It also supports both CPU and GPU for training.

17.6 Regression Imputation

The resource I used mentions this under the imputation for categorical variables, unless they treat each category as a **regressor variable**(numeric) but in my humble opinion, I think it can be used as a classification (in case we want to predict a category rather than a numeric outcome)? Regression Imputation consists of using some selected prediction of a missing value on a variable of interest. For instance, to predict the missing value for the variable, say X_1, use this variable as a function of other variables, say X_2 and X_3 in a model that could even include the dependent variable Y

Cons: The **uncertainty** is not incorporated very well because the estimates are random variables.

17.7 Imputation Using Interpolation

I think this might fit if the data is time series but I'm not sure if it can be used in non-time series data Suppose a varoable X is measured at times $t = 1, 2, 3(X_1, X_2, and X_3)$ and some values are missing at $t = 2(X_2)$, then $X_2 = \frac{X_1+X_3}{2}$

Multiple Imputation

Single based Imputation Methods mentioned earlier are an improvement over the case deletion method, but they do not account for the uncertainty in the imputations as imputed values are treated as true rather than estimates of the missing values leading to the under estimation of the variance of the estimates and the distortion of relationships among variables.Goal of **multiple imputation** is to account for uncertainty in imputed values. This method uses a selected model, such as a regression model to predict missing values on a variable. Instead of picking one value for the missing value, many values are chosen and the uncertainty is presented in the variance covariance matrix.

17.8 Multivariate Normal Imputation (MNVI)

- MNVI assumes all variables in the model are normally distributed problem for us?
- uses Markov Chain Monte Carlo Procedure to obtained imputed values from estimated multivariate distribution allowing uncertainty

17.9 Multiple Imputation by Chained Equations (MICE)

This type of imputation works by filling the missing data multiple times. The chained equations approach is also very flexible and can handle different variables of different data types (ie., continuous or binary) as well as complexities such as bounds or survey skip patterns.

Other Imputation Methods

17.10 Stochastic regression imputation:

It is quite similar to regression imputation which tries to predict the missing values by regressing it from other related variables in the same dataset plus some random residual value.

17.11 Extrapolation and Interpolation:

It tries to estimate values from other observations within the range of a discrete set of known data points.

17.12 Hot-Deck Imputation

Works by randomly choosing the missing value from a set of related and similar variables.

17.13 Some papers for handling missing categorical data (did not read them at length – just the abstract)

- Combining instance selection for better missing value imputation
- Missing data imputation using decision trees and fuzzy clustering with iterative learning
- Probabilistic neural network based categorical data imputation

References

- `https://projecteuclid.org/download/pdfview_1/euclid.bjps/1481619615`
- `http://www.jds-online.com/files/JDS-612.pdf`
- `https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779`
- `https://github.com/awslabs/datawig`
- `https://medium.com/analytics-vidhya/ways-to-handle-categorical-column-missing-data-its-implementations-15dc4a56893`

18 Advanced ML Evaluation Techniques

In this paper, authors seek to solve the problem of predicting students who are at risk of dropping out of highschool or not. They have a **binary classification problem** and they predict if the student graduates (0) or does not (1). School principals want to see if they can have enough budget to afford those - **at an**

early stage - that are at risk of not graduating and help them with additional materials. Non-Machine learning practitioners do not value regular evaluation metrics like precision and recall because it means nothing to them, they want some evaluation techniques that could help understand the predictions more from their point of view

Disclaimer

- I have read only sections 5 & 6 from the paper as instructed by Dr. Fatima
- I have found a github repository for their work here: https://github.com/dssg/student-early-warning but their work is not complete at all, the only thing that they did - with regards to sections 5 & 6, is compute the ***risk***. The rest of the work is done solely by me.
- I used the data they provided in their github repository to complete my work, they also do not provide the complete data and there are differences between the column names mentioned in the paper and the columns in the dataset they provide. But I still worked with this incomplete version of the data

18.1 Analysis of Predictive Models

School districts are interested in identifying those students who are are at risk of not graduating high school on time so that plan their resource allocation ahead of time.
Goal: predict if a student is at **risk** of not graduating high school on time

18.2 Evaluation Using Traditional Metrics

- We have a binary outcome, use standard evaluation metrics such as *accuracy, precision, recall, f-measure, and AUC*
- Their data is **imbalanced**:
  1. **Class 0: 91.4%**
  2. **Class 1: 8.6 %**
- I must do a **stratified** split of classes between training and testing such that if the original data has X% 0s and Y% 1s, then the training as well as the testing must also have X% 0s and Y% 1s
- I have added **SMOTE** so that we can **oversample** our **training** data before predicting
- I used MinMax Scaling (just for now because my focus is on the implementation of the ideas in sections 5 & 6)

- In this paper they only use shallow models, this what I also do (just for now because my focus is on the implementation of the ideas in sections 5 & 6)
- In Figure ?? below, I show the ROC curves of various models

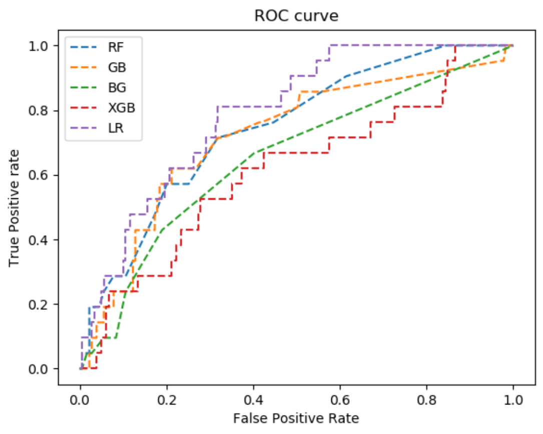


Figure 5: ROC Curves of various shallow models trained on the data provided

18.3 Problem with Standard Evaluation Metrics

1. Educators think about the performance of an algorithm in a slightly different fashion
2. the availability of resources varies with time. For example, a school might support 100 students in 2012, and 75 in 2013.
3. We want to build algorithms that can cater to these changing settings

18.4 Solution: Risk Estimates

- an algorithm, to cater to their needs, must provide them with a list of students ranked by some measure of *risk* such that, the students at the top of the list are at a higher risk of not graduating on time.
- Once educators have such a ranked list, they can choose the **top k** students from it and provide assistance to them
- **Challenge:** We only have binary records, but fortunately, all classification algorithms provide internally a probability of each class we have, we can use this probability to derive our **risk** measure

18.5 Ensuring Quality Of Risk Estimates

18.5.1 From Models to Risk Estimates

- The output of our predictions are binary: either 0 (will graduate on time) or 1 (will not graduate on time)
- Algorithms allow us to get the **probability** of each class. Here, we focus on the probability of the **positive** class (which shows the probability of not graduating on time). We use the probability of the predictions on the **final testing data**

18.5.2 Measuring the Goodness of Risk Scores

- We first get the probability of not graduating on time (class: 1) for each of the testing instances (students) we have. We call this ***risk***
- Rank the testing instances (students) in descending order of **risk** estimates. This way, students with higher probability of not graduating on time are at the top of the list.
- Group students into **bins (percentiles)** based on their risk scores. We can decide, for example, to choose 10 bins, and when we do, then the students who fall between the 10th and 20th percentile have very little risk scores, those between 20th and 30th have more risk scores than the latter
- My problem with this is as follows: we are doing the 3 bullets above for all the algorithms we have. The distribution of risk scores, for un-balanced classification problems, is usually very skewed and therefore, we can have the following 2 scenarios:
  - We might not be able to group them into 10 bins for example because each bin must have the same number of instances and that might not be possible in the un-balanced skewed distribution
  - A possible solution to the problem above is to decrease the number of bins.
  - Even if we decrease the number of bins, we might not be able to attain the same number of bins for every algorithm we have. If you look at Figure 2 in their paper, you see that for all algorithms they were able to distribute risk scores into 10 bins, but in case one is not able, we cannot produce such a plot
  - One solution is to distribute them into bins but **without guaranteeing the same number of instances in each bin**
  - Even if we are able to produce such a plot, the 10 bins won't be unique (in terms of lower and upper limits) for each algorithm. Is that valid to work with ?

- SOLUTION: I fixed this problem. I enforced the same number of items to happen in each bin **according to the number of predictions**. Number of items per bin = $numberofpredictions//nbbins$. The // in python is an **integer division** operator to divide two numbers and round their quotient down to nearest integer. This will make sure that all bins have the same number of instances (except for 1 bin if it happens that the $numberofpredictions$ and $nbbins$ are not divisible)
- For each bin, we produce the **mean empirical risk** which is the fraction of students, from that bin, who actually (as per ground truth) fail to graduate on time.
- This curve is called **mean empirical risk** curve
- An algorithm is considered to be producing good predictions if its empirical risk curve is **monotonically non-decreasing** (as the risk scores increase, we have more **correct** predictions about the positive class (more students who actually - as per ground truth - fail to graduate on time))

I present below the empirical risk curve of the shallow models I trained on - **with incorporating SMOTE** and MinMax Scaling

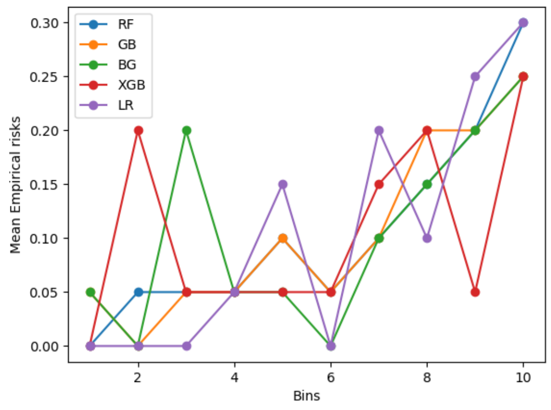


Figure 6: Mean Empirical Risks per Model

18.5.3 Comparative Evaluation of Risk Estimates

It is good to see the models performances on the **Top K** students **at risk**. The steps we do to attain this:

1. Rank Testing instances (students) by their **risk** scores
2. Define a list of **K**s

3. For each value **k** of **K**:
   (a) get the top **k** predicted values
   (b) get the ground truth of these **top k**
   (c) compute the precision/recall

We get the precision and recall at top K and we produce the following curves:

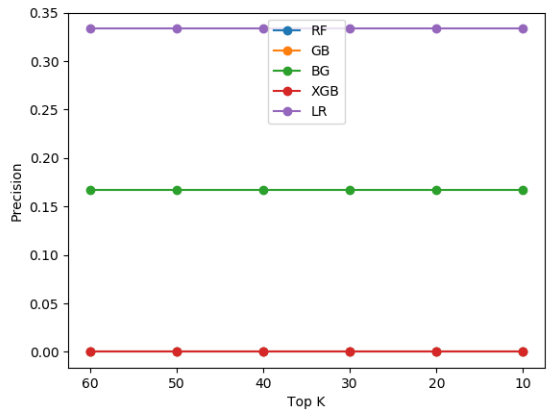


Figure 7: Precision at Top K

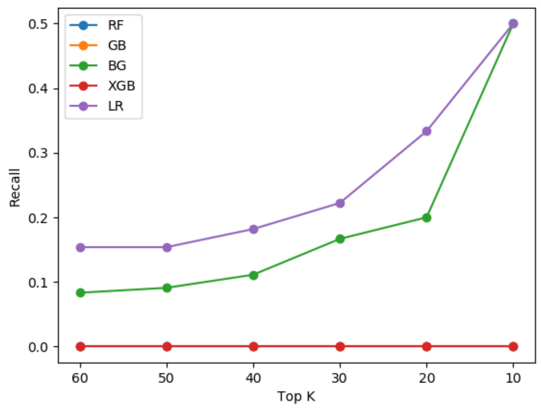


Figure 8: Recall at Top K

18.6 Interpreting Classifier Outputs - FP Growth

Frequent Patterns

To understand FP Growth algorithm, we need to first understand association rules.

Association Rules uncover the relationship between two or more attributes. It is mainly in the form of- If antecedent than consequent. For example, a supermarket sees that there are 200 customers on Friday evening. Out of the 200 customers, 100 bought chicken, and out of the 100 customers who bought chicken, 50 have bought Onions. Thus, the association rule would be- If customers buy chicken then buy onion too, with a support of 50/200 = 25% and a confidence of 50/100=50%.

References: https://www.mygreatlearning.com/blog/understanding-fp-growth-algorithm/

18.7 Technicalities

Usually, for identifying frequent patterns, we are represented with a **transactions** dataset, where each row represents **a set of items** bought by a customer. **However, we do not have transaction, we rather have multivariate dataset with a bunch of numerical columns. How can we do it?**

Disclaimer: The methodology below presents my own solution of the problem, I am not able to find such a technique anywhere on the internet.

In order to **present** values as items:

1. Consider Each column value, in a row, as being an **item** bought by the customer.
2. Our values are numeric, therefore, to avoid redundancy, I categorize the values as follows:
   - value <= 25th percentile
   - 25th percentile < value <= 75th percentile
   - value > 75th percentile
3. Do the categorization above, **per column**, for all values in the dataset
4. Achieve the so called "Item Dataset" where each row has one item (from the categories generated above) per column
5. Apply the FP-growth Technique on the generated "Item Dataset"
6. Extract the most frequent patterns

The reason I did the aforementioned methodology above is because, in the **paper we are referencing (montogemery)**, they have frequent patterns like (GPA > 2.0) and (Absence rate <= 0.1), therefore I deduced that these can be taken by doing 'quartiles' of the distribution of values we have per column.

18.8 Characterizing Prediction Mistakes

1. Get all frequent patterns using the methodology I created above that incorporates FP-Growth
2. Rank predictions based on risk score probabilistic estimates
3. Create a new field called *mistake* which is 1 if the prediction does not match ground truth and 0 otherwise
4. **For each frequent pattern** identify the **probability of mistake** by computing the number of mistakes done per frequent pattern
5. Do all of the above bullets **for each model. Therefore, we end up with a probability of mistake for each frequent pattern, per model**

18.9 Comparing Classifier Predictions

When we present educators with a suite of algorithms, they are keen on understanding the differences between *rank orderings* produced by each of these algorithms.
We will be using **Jaccard Similarity** for measuring similarity between predictions, **also at Top K**:

Let's say we have 200 testing instances (students). One model might put the highest risk score for the 180th testing instance, another model might put the highest risk score for the 190th testing instance. Therefore, it is important to note that in this exercise, we find which testing instances were chosen to be in the top k between model 1 and model 2, and we compute, using the jaccard similarity, the intersection of these testing instances (over their union). Good models must have very high intersections if they are classifying instances properly.

1. Get all possible **combinations** of model pairs (we are using a suite of models)
2. For each model pair, and for each **k** in **K**:
   (a) get the testing instances at top **k** from model 1 of the model pair
   (b) get the testing instances at top **k** from model 2 of the model pair
   (c) Compute jaccard similarity as the intersection of the two testing instances from each model over their union

After doing this, we get the following results:

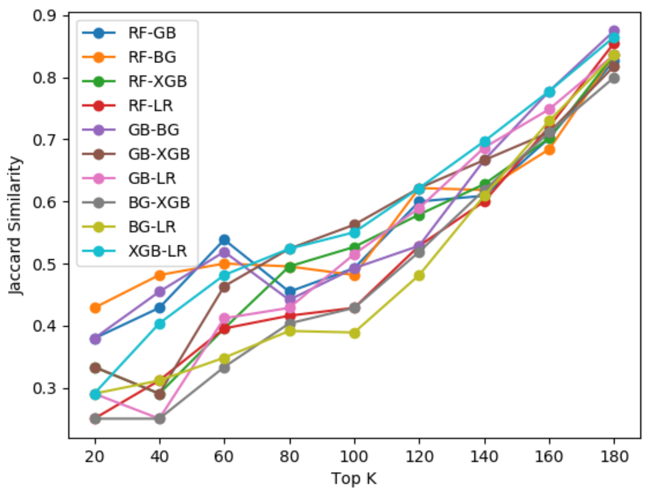


Figure 9: Jaccard Similarity of students at risk for various algorithms

19 Model Agnostic Meta Learning (MAML by Chelsea Finn)

MAML is well suited for small datasets because it **learns to learn**, from the very few examples it starts training on, and each time it encounters new examples it learns from them.

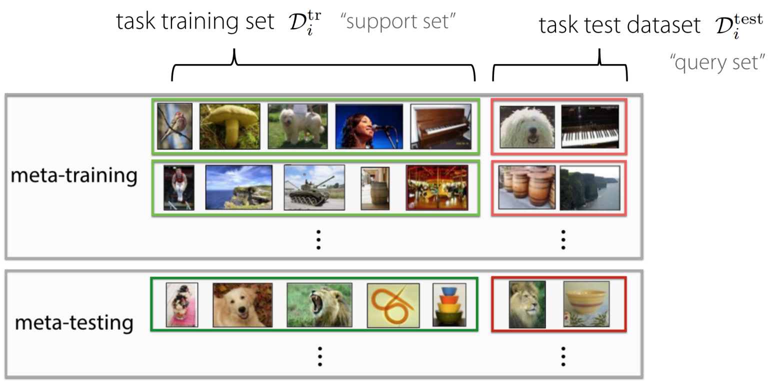
19.1 Code Modifications

In order to use MAML, we hav eto modify the code for MAML found on github by chelsea finn et al. The url for the github: `https://github.com/cbfinn/maml`

Data Generation Process

In usual deep learning exercises, we have the notion of **batches**. Each **batch** contains a number of samples of the data by which the model trains on. We can control the number of batches when we build the model's architecture and it is usually something we **tune**.
In **Meta Learning**, we have the notion of **episodes**.

Some terminology


task training set $\mathcal{D}_i^{\mathrm{tr}}$
"support set"
task test dataset $\mathcal{D}_i^{\mathrm{test}}$
"query set"
meta-training
meta-testing

k-shot learning: learning with **k** examples per class (or **k** examples total for regression)

N-way classification: choosing between N classes

Question: What are k and N for the above example? (answer in chat)

In general, each **episode** must contain **K** samples from each class found in the **episode**.

We say **K-way N-shot** classification, meaning that our problem is to predict the correct class among **K** classes, using **N** shots per class.

We will find in the literature that the number of **examples/samples** to include per class is a parameter to tune.

- In the MAML paper, the authors were trying to predict the correct characters using the **omniglot** dataset (which is a dataset that includes 1623 characters from 50 different alphabets).
- There exist 20 instances of each character
- the authors tried **20-way 1-shot** and **5-way 1-shot** and the former yielded better results

Support and Query sets

Each episode contains a support and query set. The support set is used as an "**inner training**" and the **query** set is used as a "**validation**" set to validate the result on the training. The model then learns from the **errors** made on the validation set in order to **optimize** its performance for the next iteration.

Notice that this happens per episode

Each of the support and query also must contain **number of samples per class**

Tuning the Number of Samples Per class in Support + Query Set

we can tune this. These parameters are often called:

- K-shot: The number of samples per class in the support set
- K-query: The number of samples per class in the Query set

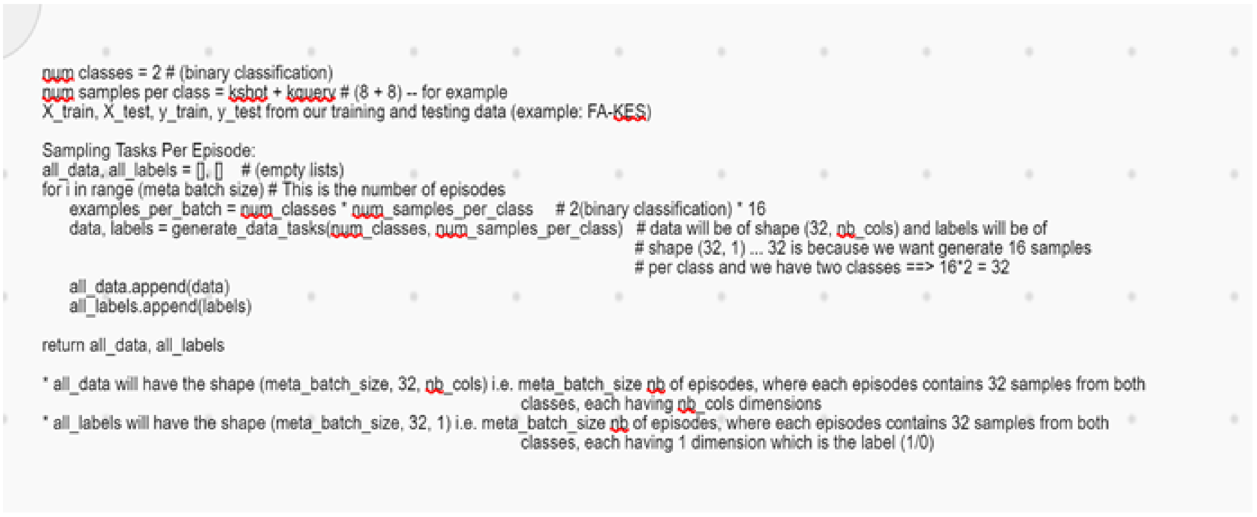
In **Finn et al**'s code on github, they did not tune this and rather always maintained the same number of **examples per class** in each of these sets.

19.2 Modification of Data Generation Process

Finn et al's code on github is made to work on images of the omniglot/Imagenet datasets.

- Modified the code to sample instances from tabular data rather than images
- Modified the code of generatimg episodes so that it also takes samples from tabular data rather than images

The below Figure shows the psuedo code of the data generation process



```
num classes = 2 # (binary classification)
num samples per class = kshot + kquery # (8 + 8) -- for example
X_train, X_test, y_train, y_test from our training and testing data (example: FA-KES)

Sampling Tasks Per Episode:
all_data, all_labels = [], []    # (empty lists)
for i in range (meta batch size) # This is the number of episodes
    examples_per_batch = num_classes * num_samples_per_class     # 2(binary classification) * 16
    data, labels = generate_data_tasks(num_classes, num_samples_per_class)   # data will be of shape (32, nb_cols) and labels will be of
                                                                              # shape (32, 1) ... 32 is because we want generate 16 samples
                                                                              # per class and we have two classes ==> 16*2 = 32
    all_data.append(data)
    all_labels.append(labels)

return all_data, all_labels

* all_data will have the shape (meta_batch_size, 32, nb_cols) i.e. meta_batch_size nb of episodes, where each episodes contains 32 samples from both
                                                classes, each having nb_cols dimensions
* all_labels will have the shape (meta_batch_size, 32, 1) i.e. meta_batch_size nb of episodes, where each episodes contains 32 samples from both
                                                classes, each having 1 dimension which is the label (1/0)
```

Figure 10: Pseudo Code of Data Generation Process

Base Models

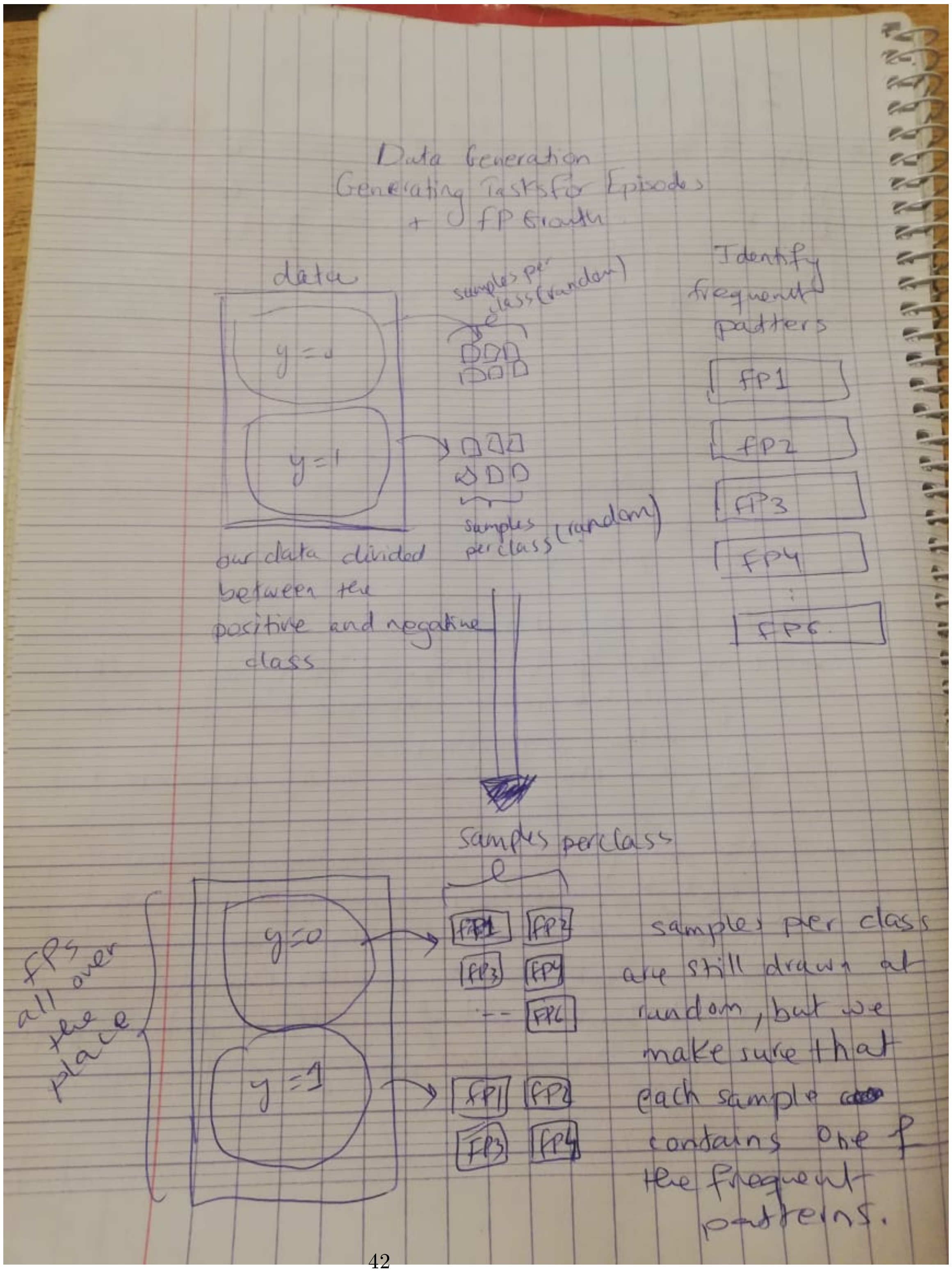
We can provide MAML with Base models from our choice (we can build the model architecture we want)

I started off with the FA-KES dataset in order to evaluate the modified MAML code on it, it is giving me overly optimistic results (near 100% accuracy) and I am just stuck fixing the bug, will communicate the results soon.

Modied MAML code on our github repository: `https://github.com/hiyamgh/dementia/tree/master/maml_finn`

19.3 Incorporating FP Growth

When we first transoformed Chelsea Finn's code on github to make it accept tabular data rather than images, we achived 60% accuracy using MAML-FNN on the FA-KES dataset whilst a suite of shallow models on FA-KES dataset achieved around 90% accuracy.
The problem was that the randomness in picking tasks for generating episodes was not passing over the whole data, and the model thus was unable to learn.
We solved this problem by incorporating FP Growth into the data generation process.



Data Generation
Generating Tasks for Episodes
+ FP Growth
data
y = 0
y = 1
our data divided between the positive and negative class
samples per class (random)
samples per class (random)
Identify frequent patters
FP1
FP2
FP3
FP4
FP6
samples per class
y=0
y=1
FP1
FP2
FP3
FP4
FP6
FPs all over the place
samples per class are still drawn at random, but we make sure that each sample contains one of the frequent patterns.

Figure 11: Data Generation Process Incorporating FP Growth

While sampling a predefined number of samples per class, we still sample tasks at random, but we make sure that each sample incorporates one of the frequent patterns, and **we ensure that each mini-batch has at least one sample from each frequent pattern found**

19.4 Results

The following table shows the results of the MAML model with a base Feed Forward neural network on the FA-KES dataset.

model	**accuracy**	**precision**	**recall**	**F1**
MAML-FNN	91.47%	88.45%	94.32%	90.86%

Table 1: Results of running MAML with baseline Feed Forward Neural Network - Standard Classification metrics

model	**AUC-ROC**	**GMEAN**	**BSS**	**F_Beta**	**PR_AUC**
MAML-FNN	92.07%	92.07%	100%	92.79%	86.42%

Table 2: Results of running MAML with baseline Feed Forward Neural Network - Cost Sensitive metrics

Previously before we incorporated **FP-Growth**, we have reached only 80% accuracy. After adding FP-growth, the accuracy has increased to 85%.It is important to note that the metrics reported above are the averages across both the **support and query sets**.

We have several hyper parameters to take into account and they are the following:

1. **num_updates**: number of inner gradient updates during training
2. **num_test_updates**: number of inner gradient updates during testing

Therefore, when we report testing results, we report it by averaging across the *num_test_updates*. In our program, we made that to be equal to 10.
It is also worth noting that, due to the randomness of data generation process (generating episodes), the testing is repeated several times. In our program, we made the number of repetitions to be 600.
Therefore when we finish testing, we end up with a **num_repeats** × **num_test_updates matrix,** were for each repeat we have num_test_updates results.
Therefore, the metrics reported above, we first do the average for each column of this matrix (we end up with a num_test_updates vector of results). Then, we report the average of that vector. This is the way MAML results are to be delivered.

20 Meta Learning Vs. Shallow Models - FAKES

20.1 Methodology

1. Full hyper parameter search for the shallow and deep models
2. Run on FAKES and compare with old results
3. Run advanced evaluation metrics on all non meta learners (done before) and new meta learners
4. We ok the procedure
5. Repeat 1 to 4 but on oversampled data and penalized models
6. Compare to Roaa's using imbalanced learning metrics and advanced metrics
7. repeat 1 to 6 using Petri Dish
8. We do shap on the ones we like the most

20.2 Quantitative Results

Without FP - Accuracy

model	accuracy	precision	recall	f1	auc
model_181	0.89205	0.82955	0.98333	0.89026	0.90909
model_37	0.86106	0.93999	0.77481	0.8471	0.86323
model_186	0.8608	0.78864	0.98333	0.86795	0.86742
model_50	0.86031	0.88333	0.83365	0.85536	0.86238
model_13	0.85147	0.91116	0.78095	0.83879	0.8534
model_43	0.85125	0.92466	0.7681	0.83644	0.85276
model_11	0.85028	0.92101	0.77117	0.83584	0.85259
model_47	0.84797	0.8824	0.80927	0.84078	0.85102
model_55	0.84606	0.8975	0.78585	0.835	0.84834
model_56	0.84579	0.86696	0.82841	0.84233	0.84818
model_36	0.84419	0.89975	0.77397	0.82969	0.84561
model_1	0.84233	0.85795	0.82596	0.83841	0.84524
model_40	0.84113	0.87993	0.79513	0.83224	0.84337
model_187	0.84091	0.79205	0.92273	0.83579	0.86818
model_42	0.84091	0.94797	0.72326	0.8187	0.84248
model_48	0.83989	0.87685	0.79279	0.83	0.84116
model_45	0.83953	0.90585	0.76016	0.82447	0.84087
model_190	0.83807	0.76098	0.96212	0.84287	0.84621
model_15	0.83794	0.92169	0.74209	0.81934	0.84007
model_14	0.83758	0.94932	0.71444	0.81325	0.83918

Table 3: Results - MAML without FP, aggregated by accuracy score

With FP - Accuracy score

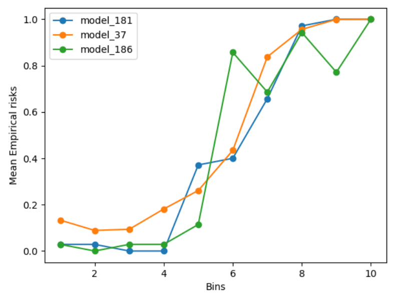
model	**accuracy**	**precision**	**recall**	**f1**	**auc**
model_616	0.93466	1	0.87992	0.93196	0.93996
model_152	0.92791	0.93887	0.91728	0.92519	0.93167
model_529	0.92756	0.91464	0.93878	0.92215	0.93371
model_682	0.9233	0.95833	0.88902	0.91829	0.92424
model_136	0.9217	0.93084	0.91161	0.91903	0.92478
model_137	0.92143	0.93472	0.90982	0.91913	0.9248
model_121	0.92001	0.94291	0.89793	0.91718	0.92305
model_118	0.91992	0.93004	0.91393	0.91818	0.92406
model_107	0.91939	0.92363	0.91577	0.91658	0.92268
model_170	0.9185	0.95976	0.87648	0.91467	0.92121
model_446	0.91832	0.96195	0.87096	0.90732	0.92213
model_109	0.9138	0.95311	0.87603	0.90722	0.91689
model_106	0.91366	0.92273	0.90359	0.91093	0.9156
model_28	0.91211	0.90241	0.92839	0.91241	0.91448
model_688	0.91193	0.88409	0.94432	0.9081	0.91932
model_29	0.91082	0.90301	0.92317	0.91048	0.91406
model_128	0.91056	0.93115	0.8888	0.90645	0.91265
model_535	0.91051	0.97781	0.81992	0.88931	0.90314
model_8	0.90874	0.92644	0.89238	0.90576	0.91311
model_134	0.90745	0.92396	0.89215	0.9048	0.91099

Table 4: Results - MAML with FP, aggregated by accuracy score

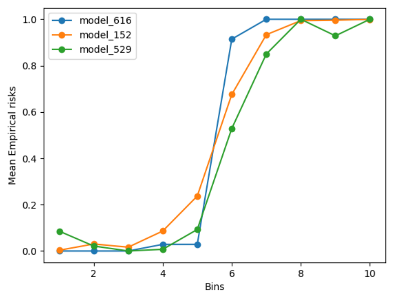
We realize that the results with FP are better.

20.3 Qualitative Results

20.4 Mean Empirical Risk Curves



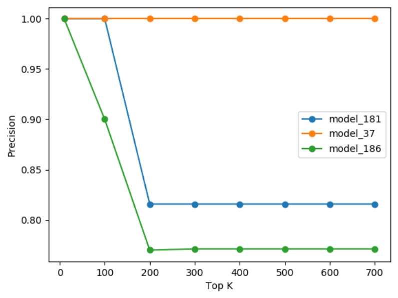
(a) Mean Empirical Risks for top 3 models - Without FP



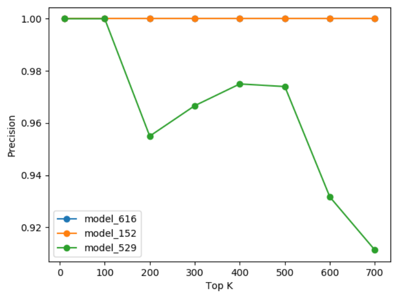
(b) Mean Empirical Risks for top 3 models - With FP

Figure 12: Mear Empirical Risks for top models - With FP

20.5 Precisions Top K



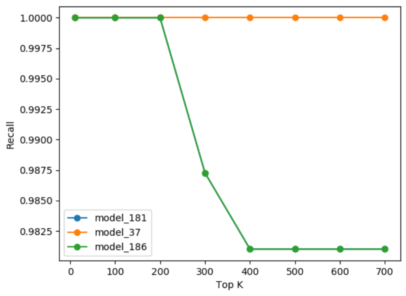
(a) Precisions at Top K for top 3 models - Without FP



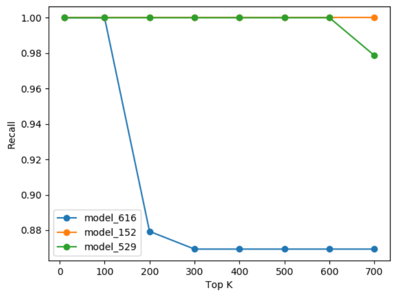
(b) Precisions at Top K for top 3 models - With FP

Figure 13: Precisions at top K for top models - With FP

20.6 Recalls Top K



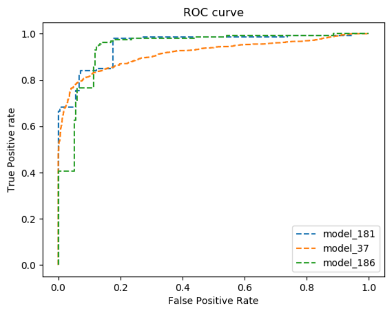
(a) Recalls at Top K for top 3 models - Without FP



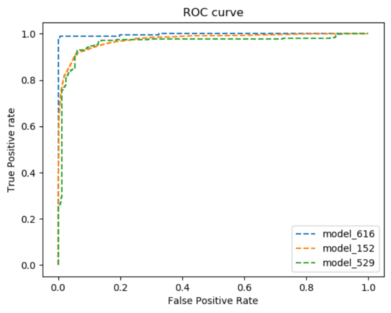
(b) Recalls at Top K for top 3 models - With FP

Figure 14: Precisions at top K for top models - With FP

20.7 ROC Curves



(a) ROC Curve for top 3 models - Without FP



(b) ROC Curve for top 3 models - With FP

Figure 15: ROC Curves top K for top models - With FP

21 Meta Learning vs Shallow Models - Dementia Dataset

21.1 Cost Sensitive Learning in Neural Networks

We have many types of cost sensitive learning, here, we will be focusing on two types which are **weighting** and **miss-classification error/cost matrix**.

It is important to note that, after the input is forwarded from the input layer to the output layer, the neural network computes the **loss** which quantifies how much mistake did it make, if any.

Key defenition - Loss

The loss function in case of regression is the **mean squared error**, whilst that for classification is the **cross entropy loss** (binary cross entropy (for binary classification tasks) or categorical cross entropy for multi-classification tasks)

Key defenition - Logits

Loss if computed on the **logits**. Logits are the raw outputs of the last layer of the neural network. Logits interpreted to be the unnormalised (or not-yet normalised) predictions (or outputs) of a model. These can give results, but we don't normally stop with logits, because interpreting their raw values is not easy.

Let's demonstrate this through an example, say we want to classify images as being either a cat or a dog. For the first new image, we get logit values out of 16.917 for a cat and then 0.772 for a dog. Higher means better, or ('more likely'), so we'd say that a cat is the answer. The correct answer is a cat, so the model worked. If we apply softmax activation to the logits, we get p(cat) = 0.99 and p(dog) = 0.0001

Weighting

Weighting is applied to the loss such that smaller weight values result in errors, i.e. less update the model coefficients and larger weight results in more errors, i.e. more updates to the model coefficients.

Multiply the logit with a weight vector representing scaling factor of each class.

Miss-Classification Errros/Cost-Matrix

The problem with class weighting is that it applies the weighting to all the data that belongs to the class, whilst we want it to depend on the miss classification error.

actual/predicted	class 0	class 1
class 0	0	0.25
class 1	0.25	0

Table 5: cost matrix of classifications/miss-classifications

The table above shows that the cost of miss-classifying class 0 as class 1 or classifying class 1 as class 0 is 0.25 while correct classifications have no cost.

21.2 Hyper Parameters

hyper parameter	**description**	**values**
Meta Learning Hyper Parameters		
meta batch size	number of tasks sampled pre meta update	4, 8, 16
meta learning rate	the base learning rate of the generator	0.1, 0.001
update batch size	number of examples used for inner gradient updates	4, 8, 16
Base Learner Hyper Parameters		
update learning rate	step size alpha for inner gradient update	0.1, 0.001
hidden layers	the number of hidden layers and their nodes	264, 128, 64 128, 64, 64 128, 64 128
activation functions	non-linear activation functions	relu, sigmoid, tanh, softmax, swish
Cost Sensitive Hyper Parameters		
weights	weights vector applied to logits	[1, 10], [1, 100], [1,1], [10, 1], [100, 1]
sampling strategy	sampling strategy used for under/over - sampling the data	minority, majority, all 0.5, 0.75, 1
FP Growth Hyper parameters		
minimum support	minumum support for frequent patterns	0.7, 0.8, 0.9

Table 6: Hyper Parameter Space

21.3 Winning Hyper Parameters

model	miter	mbs	mlr	ulr	dh	afn	nu	ifp	fp_supp	weights	sampling_strategy
model_134	1000	16	0.1	0.1	128, 64	relu	4	1	0.9	1, 100	not minority
model_120	1000	16	0.1	0.1	128, 64	relu	4	1	0.8	100, 1	0.75
model_226	1000	16	0.1	0.1	128	relu	4	1	0.9	1, 100	0.5
model_186	1000	16	0.1	0.1	128	relu	4	1	0.8	1, 1	0.75
model_194	1000	16	0.1	0.1	128	relu	4	1	0.8	1, 100	not minority
model_149	1000	16	0.1	0.1	128, 64	relu	4	1	0.9	100, 1	1
model_238	1000	16	0.1	0.1	128	relu	4	1	0.9	100, 1	0.5
model_115	1000	16	0.1	0.1	128, 64	relu	4	1	0.8	100, 1	minority
model_205	1000	16	0.1	0.1	128	relu	4	1	0.8	100, 1	minority
model_185	1000	16	0.1	0.1	128	relu	4	1	0.8	1, 1	1
model_228	1000	16	0.1	0.1	128	relu	4	1	0.9	1, 100	0.75
model_60	1000	16	0.1	0.1	128, 64, 64	relu	4	1	0.9	100, 1	0.75
model_486	1000	16	0.1	0.001	128	relu	4	1	0.9	1, 1	0.75
model_484	1000	16	0.1	0.001	128	relu	4	1	0.9	1, 1	0.5
model_201	1000	16	0.1	0.1	128	relu	4	1	0.8	10, 1	all
model_191	1000	16	0.1	0.1	128	relu	4	1	0.8	1, 10	1
model_190	1000	16	0.1	0.1	128	relu	4	1	0.8	1, 10	0.5
model_22	1000	16	0.1	0.1	128, 64, 64	relu	4	1	0.8	10, 1	0.5
model_481	1000	16	0.1	0.001	128	relu	4	1	0.9	1, 1	minority
model_116	1000	16	0.1	0.1	128, 64	relu	4	1	0.8	100, 1	not minority
model_209	1000	16	0.1	0.1	128	relu	4	1	0.8	100, 1	1

Table 7: Winning Hyper Parameters - MAML with FP, using top 10 columns

model	miter	mbs	mlr	ulr	dh	afn	nu	ifp	fp_supp	weights	sampling_strategy
model_35	1000	16	0.1	0.1	128, 64, 64	relu	4	1	0.9	1, 1	1
model_215	1000	16	0.1	0.1	128	relu	4	1	0.9	1, 1	1
model_234	1000	16	0.1	0.1	128	relu	4	1	0.9	10, 1	0.75
model_54	1000	16	0.1	0.1	128, 64, 64	relu	4	1	0.9	10, 1	0.75
model_226	1000	16	0.1	0.1	128	relu	4	1	0.9	1, 100	0.5
model_49	1000	16	0.1	0.1	128, 64, 64	relu	4	1	0.9	10, 1	minority
model_213	1000	16	0.1	0.1	128	relu	4	1	0.9	1, 1	all
model_51	1000	16	0.1	0.1	128, 64, 64	relu	4	1	0.9	10, 1	all
model_219	1000	16	0.1	0.1	128	relu	4	1	0.9	1, 10	all
model_36	1000	16	0.1	0.1	128, 64, 64	relu	4	1	0.9	1, 1	0.75
model_37	1000	16	0.1	0.1	128, 64, 64	relu	4	1	0.9	1, 10	minority
model_217	1000	16	0.1	0.1	128	relu	4	1	0.9	1, 10	minority
model_451	1000	16	0.1	0.001	128	relu	4	1	0.8	1, 1	minority
model_41	1000	16	0.1	0.1	128, 64, 64	relu	4	1	0.9	1, 10	1
model_229	1000	16	0.1	0.1	128	relu	4	1	0.9	10, 1	minority
model_96	1000	16	0.1	0.1	128, 64	relu	4	1	0.8	1, 1	0.75
model_235	1000	16	0.1	0.1	128	relu	4	1	0.9	100, 1	minority
model_94	1000	16	0.1	0.1	128, 64	relu	4	1	0.8	1, 1	0.5
model_211	1000	16	0.1	0.1	128	relu	4	1	0.9	1, 1	minority
model_220	1000	16	0.1	0.1	128	relu	4	1	0.9	1, 10	0.5

Table 8: Winning Hyper Parameters - MAML with FP, using top 20 columns

21.4 Quantitative Results - Without FP Growth

Without FP - Top 10 - F2 score

model	f2	gmean	bss	pr_auc	sensitivity	specificity	ppv	accuracy	precision	recall	f1	auc
model_352	0.90333	0.75375	0.49756	0.69441	0.98379	0.52678	0.666052	0.76119	0.69704	0.98379	0.81014	0.75528
model_345	0.89287	0.78217	0.55922	0.71829	0.95405	0.61153	0.683306	0.78533	0.72704	0.95405	0.81887	0.78279
model_320	0.89128	0.75709	0.50194	0.69848	0.96402	0.55212	0.639025	0.75959	0.70519	0.96402	0.80702	0.75807
model_14	0.86021	0.79632	0.59837	0.73939	0.89115	0.70184	0.463194	0.79901	0.76344	0.89115	0.81953	0.79649
model_339	0.84859	0.57341	0.14306	0.57221	0.98096	0.17565	0.795358	0.58896	0.57544	0.98096	0.71481	0.57831
model_322	0.8479	0.62023	0.22292	0.5938	0.96255	0.28362	0.779757	0.62713	0.59752	0.96255	0.72774	0.62308
model_350	0.84533	0.53957	0.08953	0.54749	0.98949	0.10127	0.918161	0.56108	0.54861	0.98949	0.69957	0.54538
model_353	0.84431	0.51309	0.04078	0.53257	0.9994	0.03937	0.936137	0.53711	0.53264	0.9994	0.68982	0.51938
model_278	0.84388	0.54139	0.09142	0.54802	0.98684	0.10649	0.903981	0.56126	0.54841	0.98684	0.69893	0.54666
model_269	0.84294	0.50619	0.03184	0.52693	1	0.02435	0.989652	0.53125	0.52693	1	0.68619	0.51217
model_338	0.84148	0.51895	0.05515	0.53486	0.99181	0.05842	0.947485	0.54297	0.53542	0.99181	0.69044	0.52512
model_224	0.84126	0.49744	0.02121	0.52272	1	0.00786	0.798795	0.52486	0.52272	1	0.68304	0.50393
model_254	0.84125	0.4996	0.01903	0.52383	1	0.01184	0.995406	0.52521	0.52383	1	0.68349	0.50592
model_236	0.8411	0.49682	0.01921	0.52261	1	0.00702	0.996349	0.52433	0.52261	1	0.68289	0.50351
model_308	0.8406	0.49501	0.0157	0.52164	1	0.0034	0.965703	0.52255	0.52164	1	0.68205	0.5017
model_349	0.84014	0.49342	0.01223	0.52085	1	0.0003	0.962425	0.52095	0.52085	1	0.68134	0.50015
model_237	0.84009	0.49327	0.01178	0.52077	1	0	0.867752	0.52077	0.52077	1	0.68127	0.5
model_240	0.84009	0.49327	0.01178	0.52077	1	0	0.957389	0.52077	0.52077	1	0.68127	0.5
model_206	0.84009	0.49327	0.01178	0.52077	1	0	1	0.52077	0.52077	1	0.68127	0.5
model_194	0.84009	0.49327	0.01178	0.52077	1	0	1	0.52077	0.52077	1	0.68127	0.5

Table 9: Results - MAML without FP, using top 10 columns - aggregated by f2 score

Without FP - Top 10 - BSS score

model	f2	gmean	bss	pr_auc	sensitivity	specificity	ppv	accuracy	precision	recall	f1	auc
model_14	0.86021	0.79632	0.59837	0.73939	0.89115	0.70184	0.463194	0.79901	0.76344	0.89115	0.81953	0.79649
model_23	0.78183	0.79804	0.58124	0.75716	0.78222	0.81416	0.485702	0.78835	0.81668	0.78222	0.78775	0.79819
model_62	0.82216	0.78818	0.57375	0.73532	0.84098	0.73562	0.542736	0.78587	0.77182	0.84098	0.79905	0.7883
model_80	0.83658	0.78878	0.56362	0.73099	0.86669	0.71138	0.498403	0.78409	0.76163	0.86669	0.80112	0.78903
model_345	0.89287	0.78217	0.55922	0.71829	0.95405	0.61153	0.683306	0.78533	0.72704	0.95405	0.81887	0.78279
model_218	0.82873	0.78292	0.55673	0.72729	0.85488	0.71134	0.553198	0.77947	0.75825	0.85488	0.79693	0.78311
model_190	0.76754	0.77943	0.55553	0.7387	0.76206	0.79693	0.477391	0.77823	0.80368	0.76206	0.77823	0.7795
model_65	0.77679	0.78629	0.55222	0.74123	0.77886	0.79402	0.494065	0.77663	0.80029	0.77886	0.77948	0.78644
model_24	0.78087	0.7782	0.55068	0.73583	0.78472	0.77192	0.511865	0.77202	0.78704	0.78472	0.77944	0.77832
model_231	0.79337	0.78033	0.54231	0.72965	0.80544	0.75552	0.487545	0.77326	0.77994	0.80544	0.78196	0.78048
model_63	0.79132	0.77831	0.54228	0.73022	0.8014	0.75547	0.51611	0.77308	0.77964	0.8014	0.78182	0.77844
model_323	0.82045	0.77649	0.54221	0.72087	0.8453	0.70837	0.517651	0.77681	0.75741	0.8453	0.79113	0.77684
model_32	0.80356	0.77484	0.54067	0.72554	0.82299	0.72715	0.549869	0.76758	0.76227	0.82299	0.78159	0.77507
model_61	0.77174	0.78218	0.5404	0.73613	0.77408	0.7907	0.484901	0.77202	0.79967	0.77408	0.7749	0.78239
model_272	0.79675	0.77015	0.53894	0.72001	0.80941	0.73107	0.546094	0.76776	0.76369	0.80941	0.7814	0.77024
model_41	0.76633	0.77684	0.53883	0.73439	0.77045	0.78366	0.488854	0.76705	0.79119	0.77045	0.7674	0.77706
model_11	0.76596	0.77365	0.53628	0.72885	0.7664	0.78105	0.501777	0.76811	0.78725	0.7664	0.7695	0.77373
model_38	0.79068	0.77115	0.53259	0.72258	0.80524	0.73732	0.52104	0.76438	0.76633	0.80524	0.77525	0.77128
model_7	0.77149	0.77105	0.53034	0.72497	0.77641	0.76595	0.505823	0.76634	0.78046	0.77641	0.76943	0.77118
model_271	0.76839	0.77274	0.5302	0.72481	0.77232	0.77349	0.508718	0.76651	0.78414	0.77232	0.76866	0.77291

Table 10: Results - MAML without FP, using top 10 columns - aggregated by bss score

Without FP - Top 20 - F2 score

model	f2	gmean	bss	pr_auc	sensitivity	specificity	ppv	accuracy	precision	recall	f1	auc
model_62	0.88704	0.84811	0.68757	0.79225	0.90992	0.78662	0.565734	0.8473	0.81941	0.90992	0.85773	0.84827
model_248	0.88558	0.84443	0.68914	0.79239	0.90777	0.78143	0.546026	0.84393	0.8192	0.90777	0.85685	0.8446
model_81	0.88504	0.83721	0.66325	0.78068	0.91293	0.76188	0.564917	0.83665	0.80574	0.91293	0.85008	0.8374
model_354	0.88221	0.75589	0.49779	0.69231	0.95133	0.56227	0.707806	0.75888	0.70073	0.95133	0.8006	0.7568
model_195	0.88128	0.67552	0.34213	0.62794	0.98837	0.36704	0.728501	0.68537	0.62911	0.98837	0.76321	0.67771
model_242	0.88092	0.85081	0.69474	0.79971	0.8988	0.80305	0.552548	0.84783	0.83053	0.8988	0.85864	0.85093
model_182	0.87589	0.85698	0.70236	0.80722	0.88865	0.82545	0.541004	0.85298	0.84212	0.88865	0.86046	0.85705
model_63	0.87559	0.84914	0.68498	0.79755	0.89226	0.80615	0.552762	0.84588	0.82936	0.89226	0.85486	0.8492
model_293	0.87481	0.71732	0.43758	0.67395	0.95351	0.48485	0.705024	0.72585	0.68072	0.95351	0.78563	0.71918
model_320	0.86844	0.63237	0.25245	0.60883	0.98702	0.28546	0.841114	0.64435	0.61169	0.98702	0.74494	0.63624
model_222	0.86748	0.86194	0.72431	0.82265	0.86954	0.8546	0.52065	0.86346	0.86676	0.86954	0.86584	0.86207
model_247	0.86722	0.83954	0.67257	0.79004	0.88369	0.79558	0.547744	0.83736	0.82477	0.88369	0.8472	0.83963
model_279	0.86589	0.7157	0.39497	0.67115	0.94754	0.48755	0.765059	0.7136	0.68122	0.94754	0.77897	0.71755
model_183	0.86566	0.85537	0.68747	0.80534	0.87736	0.83363	0.538794	0.84677	0.84429	0.87736	0.85335	0.85549
model_66	0.8655	0.8623	0.71384	0.82053	0.86999	0.85484	0.528659	0.85955	0.86621	0.86999	0.86226	0.86242
model_269	0.86433	0.59075	0.18055	0.57252	1	0.18879	0.913954	0.60671	0.57252	1	0.72303	0.5944
model_80	0.86431	0.81689	0.62899	0.76174	0.89019	0.744	0.554633	0.81641	0.7919	0.89019	0.83211	0.81709
model_79	0.86278	0.8117	0.61649	0.75751	0.8905	0.73349	0.562066	0.81197	0.78795	0.8905	0.82901	0.812
model_61	0.86227	0.83379	0.65969	0.78426	0.87825	0.78946	0.55089	0.8315	0.81838	0.87825	0.84251	0.83386
model_68	0.86179	0.82319	0.62727	0.76707	0.88683	0.76002	0.546381	0.818	0.80062	0.88683	0.83261	0.82342

Table 11: Results - MAML without FP, using top 20 columns - aggregated by f2 score

Without FP - Top 20 - BSS score

model	f2	gmean	bss	pr_auc	sensitivity	specificity	ppv	accuracy	precision	recall	f1	auc
model_306	0.85542	0.86631	0.72916	0.8324	0.85023	0.88245	0.500059	0.86577	0.88414	0.85023	0.8646	0.86634
model_186	0.84994	0.8671	0.72596	0.83355	0.84288	0.89139	0.496716	0.86275	0.8892	0.84288	0.86255	0.86714
model_222	0.86748	0.86194	0.72431	0.82265	0.86954	0.8546	0.52065	0.86346	0.86676	0.86954	0.86584	0.86207
model_66	0.8655	0.8623	0.71384	0.82053	0.86999	0.85484	0.528659	0.85955	0.86621	0.86999	0.86226	0.86242
model_182	0.87589	0.85698	0.70236	0.80722	0.88865	0.82545	0.541004	0.85298	0.84212	0.88865	0.86046	0.85705
model_242	0.88092	0.85081	0.69474	0.79971	0.8988	0.80305	0.552548	0.84783	0.83053	0.8988	0.85864	0.85093
model_248	0.88558	0.84443	0.68914	0.79239	0.90777	0.78143	0.546026	0.84393	0.8192	0.90777	0.85685	0.8446
model_246	0.84633	0.84889	0.68776	0.80478	0.84733	0.85055	0.515179	0.8457	0.85535	0.84733	0.8473	0.84894
model_62	0.88704	0.84811	0.68757	0.79225	0.90992	0.78662	0.565734	0.8473	0.81941	0.90992	0.85773	0.84827
model_183	0.86566	0.85537	0.68747	0.80534	0.87736	0.83363	0.538794	0.84677	0.84429	0.87736	0.85335	0.85549
model_234	0.84258	0.84682	0.68516	0.80464	0.84131	0.85239	0.509578	0.84428	0.85499	0.84131	0.84586	0.84685
model_63	0.87559	0.84914	0.68498	0.79755	0.89226	0.80615	0.552762	0.84588	0.82936	0.89226	0.85486	0.8492
model_12	0.82667	0.84397	0.67849	0.8078	0.82001	0.86805	0.491581	0.83842	0.86685	0.82001	0.83907	0.84403
model_252	0.83793	0.84727	0.677	0.80229	0.83685	0.8578	0.516448	0.8418	0.85514	0.83685	0.84199	0.84732
model_189	0.8504	0.84771	0.67656	0.80107	0.85714	0.83847	0.52129	0.83931	0.84569	0.85714	0.8447	0.8478
model_24	0.83182	0.8409	0.67474	0.79966	0.82969	0.85223	0.507776	0.83718	0.85535	0.82969	0.83768	0.84096
model_190	0.84424	0.84315	0.67455	0.8004	0.84644	0.83999	0.507351	0.83949	0.84764	0.84644	0.84317	0.84321
model_215	0.86131	0.84568	0.67315	0.79907	0.87277	0.8189	0.544643	0.84073	0.839	0.87277	0.84896	0.84584
model_247	0.86722	0.83954	0.67257	0.79004	0.88369	0.79558	0.547744	0.83736	0.82477	0.88369	0.8472	0.83963
model_244	0.82713	0.84247	0.66967	0.80195	0.82362	0.86153	0.500601	0.83629	0.86312	0.82362	0.8363	0.84258

Table 12: Results - MAML without FP, using top 20 columns - aggregated by bss score

21.5 Insights

1. We realize that the highest F2 score reached was by an MAML model, that **did not** incorporate FP Growth, and taking the top 10 columns selected by feature selection. **The highest F2 score is 0.9 shown in Table 9**

2. We realize that the highest BSS score reached by an MAML model, that **did not** incorporate FP Growth, and taking the top 20 columns selected by feature selection. The highest BSS score is 0.72 shown in Table 12

3. the model named *'model_306'* found in table 12 seems to be a perfect model because, although not the highest in f2 score, it has an F2 score of 0.85, a GMEAN score of 0.86, a PR_AUC score of 0.83, and a BSS score of 0.72. Therefore, this model is doing equivalently well in performance with regards to predicting the positive and the negative class.

21.6 Quantitative Results - With FP Growth

With FP - Top 10 - F2 score

model	**f2**	**gmean**	**bss**	**pr_auc**	**sensitivity**	**specificity**	**ppv**	**accuracy**	**precision**	**recall**	**f1**	**auc**
model_134	0.89145	0.82404	0.61822	0.74744	0.93859	0.70986	0.505566	0.81907	0.76279	0.93859	0.83311	0.82423
model_120	0.89068	0.77267	0.51849	0.69895	0.95991	0.58638	0.59205	0.76989	0.70631	0.95991	0.80835	0.77315
model_226	0.88507	0.76232	0.49967	0.68916	0.95843	0.56735	0.572628	0.76101	0.70081	0.95843	0.7996	0.76289
model_186	0.87592	0.83823	0.65355	0.76927	0.90272	0.77388	0.547636	0.83416	0.7971	0.90272	0.84166	0.8383
model_194	0.87164	0.76118	0.49973	0.69689	0.93293	0.59025	0.545977	0.76048	0.71086	0.93293	0.79954	0.76159
model_149	0.8663	0.76698	0.49803	0.70689	0.92768	0.60739	0.551255	0.76083	0.72535	0.92768	0.79886	0.76753
model_238	0.86621	0.8065	0.58353	0.73338	0.9042	0.70914	0.476054	0.80291	0.75933	0.9042	0.81912	0.80667
model_115	0.85934	0.81025	0.58698	0.73962	0.89813	0.72285	0.491688	0.80398	0.76223	0.89813	0.8127	0.81049
model_205	0.85403	0.79717	0.56475	0.72351	0.89477	0.69995	0.458266	0.79208	0.74918	0.89477	0.80397	0.79736
model_185	0.84878	0.80981	0.60571	0.74549	0.87137	0.74837	0.53602	0.81001	0.7751	0.87137	0.81835	0.80987
model_228	0.8464	0.79186	0.56245	0.72506	0.8905	0.69382	0.470074	0.78817	0.74995	0.8905	0.79499	0.79216
model_60	0.84093	0.80631	0.58013	0.74004	0.87522	0.73799	0.482142	0.80007	0.76938	0.87522	0.80155	0.8066
model_486	0.83497	0.80323	0.57238	0.73452	0.86439	0.74242	0.543126	0.79705	0.77212	0.86439	0.80204	0.80341
model_484	0.82986	0.79325	0.56278	0.72448	0.85508	0.73158	0.517505	0.78853	0.76057	0.85508	0.79866	0.79333
model_201	0.82811	0.81948	0.62763	0.76261	0.83662	0.80243	0.473389	0.81729	0.81262	0.83662	0.81878	0.81953
model_191	0.82611	0.79942	0.58545	0.73559	0.8434	0.75553	0.484182	0.80043	0.77336	0.8434	0.80343	0.79947
model_190	0.824	0.81354	0.61005	0.75153	0.83283	0.79429	0.480385	0.81268	0.80124	0.83283	0.81326	0.81356
model_22	0.82372	0.8188	0.61585	0.7583	0.83156	0.80613	0.483301	0.81463	0.80751	0.83156	0.81482	0.81884
model_481	0.82284	0.78044	0.53682	0.71357	0.85092	0.71019	0.527098	0.77646	0.7434	0.85092	0.78765	0.78055
model_116	0.8181	0.71567	0.386	0.65019	0.8892	0.54342	0.525363	0.70934	0.6758	0.8892	0.73907	0.71631

Table 13: Results - MAML with FP, using top 10 columns - aggregated by f2 score

With FP - Top 10 - BSS score

model	f2	gmean	bss	pr_auc	sensitivity	specificity	ppv	accuracy	precision	recall	f1	auc
model_186	0.87592	0.83823	0.65355	0.76927	0.90272	0.77388	0.547636	0.83416	0.7971	0.90272	0.84166	0.8383
model_201	0.82811	0.81948	0.62763	0.76261	0.83662	0.80243	0.473389	0.81729	0.81262	0.83662	0.81878	0.81953
model_218	0.80849	0.81737	0.62192	0.76403	0.80887	0.82592	0.448325	0.81499	0.82517	0.80887	0.81071	0.81739
model_134	0.89145	0.82404	0.61822	0.74744	0.93859	0.70986	0.505566	0.81907	0.76279	0.93859	0.83311	0.82423
model_22	0.82372	0.8188	0.61585	0.7583	0.83156	0.80613	0.483301	0.81463	0.80751	0.83156	0.81482	0.81884
model_21	0.8133	0.81418	0.61077	0.75814	0.81774	0.81065	0.482362	0.81179	0.811	0.81774	0.80949	0.8142
model_190	0.824	0.81354	0.61005	0.75153	0.83283	0.79429	0.480385	0.81268	0.80124	0.83283	0.81326	0.81356
model_185	0.84878	0.80981	0.60571	0.74549	0.87137	0.74837	0.53602	0.81001	0.7751	0.87137	0.81835	0.80987
model_111	0.79346	0.81067	0.60506	0.75677	0.79205	0.82937	0.473848	0.80646	0.8168	0.79205	0.79862	0.81071
model_213	0.80168	0.80371	0.59362	0.74824	0.80381	0.80368	0.480172	0.8022	0.80671	0.80381	0.80108	0.80374
model_39	0.79414	0.80202	0.58895	0.74825	0.79409	0.80999	0.477181	0.79989	0.80838	0.79409	0.79685	0.80204
model_115	0.85934	0.81025	0.58698	0.73962	0.89813	0.72285	0.491688	0.80398	0.76223	0.89813	0.8127	0.81049
model_121	0.8003	0.80413	0.58632	0.74735	0.80679	0.80159	0.499812	0.79918	0.80024	0.80679	0.79534	0.80419
model_114	0.79854	0.80144	0.58632	0.74301	0.80059	0.80233	0.494492	0.79918	0.79917	0.80059	0.7971	0.80146
model_191	0.82611	0.79942	0.58545	0.73559	0.8434	0.75553	0.484182	0.80043	0.77336	0.8434	0.80343	0.79947
model_238	0.86621	0.8065	0.58353	0.73338	0.9042	0.70914	0.476054	0.80291	0.75933	0.9042	0.81912	0.80667
model_234	0.7988	0.79974	0.5803	0.7413	0.80715	0.79243	0.460926	0.79812	0.79424	0.80715	0.79159	0.79979
model_60	0.84093	0.80631	0.58013	0.74004	0.87522	0.73799	0.482142	0.80007	0.76938	0.87522	0.80155	0.8066
model_454	0.80453	0.80398	0.57951	0.74349	0.81354	0.79458	0.498237	0.80007	0.79733	0.81354	0.79617	0.80406
model_7	0.79551	0.80342	0.5771	0.74279	0.79719	0.80971	0.482822	0.79972	0.80152	0.79719	0.79539	0.80345

Table 14: Results - MAML with FP, using top 10 columns - aggregated by bss score

With FP - Top 20 - F2 score

model	f2	gmean	bss	pr_auc	sensitivity	specificity	ppv	accuracy	precision	recall	f1	auc
model_35	0.88339	0.88793	0.76578	0.85218	0.88299	0.89292	0.505276	0.88406	0.89551	0.88299	0.88596	0.88796
model_215	0.87711	0.8784	0.74115	0.83454	0.87953	0.87732	0.517793	0.87518	0.87699	0.87953	0.87527	0.87842
model_234	0.87664	0.87315	0.74385	0.83544	0.87891	0.8675	0.521648	0.87305	0.88027	0.87891	0.87557	0.87321
model_54	0.87615	0.87149	0.73155	0.82762	0.88201	0.86107	0.521293	0.86701	0.86871	0.88201	0.87037	0.87154
model_226	0.87238	0.73393	0.47523	0.6986	0.94252	0.52822	0.541673	0.74254	0.71737	0.94252	0.79681	0.73537
model_49	0.87003	0.87359	0.74059	0.83558	0.86997	0.87724	0.49026	0.87145	0.88043	0.86997	0.87204	0.87361
model_213	0.86712	0.87315	0.73672	0.83865	0.86648	0.87993	0.501388	0.86914	0.88384	0.86648	0.87059	0.8732
model_51	0.86702	0.85519	0.69408	0.80569	0.87564	0.8348	0.514519	0.85174	0.84495	0.87564	0.8566	0.85522
model_219	0.86477	0.85635	0.70387	0.81352	0.87298	0.83992	0.520798	0.85121	0.85694	0.87298	0.85716	0.85645
model_36	0.86405	0.87819	0.74672	0.84604	0.8589	0.89754	0.492165	0.87393	0.8953	0.8589	0.87369	0.87822
model_37	0.86253	0.86849	0.7251	0.82838	0.86191	0.87512	0.496969	0.86506	0.87567	0.86191	0.86537	0.86851
model_217	0.86165	0.78094	0.56608	0.72014	0.90076	0.66173	0.616071	0.78711	0.74336	0.90076	0.81126	0.78125
model_451	0.86164	0.80377	0.52345	0.74083	0.90568	0.70316	0.600992	0.79155	0.76372	0.90568	0.81389	0.80442
model_41	0.86115	0.85642	0.7005	0.81025	0.86588	0.84702	0.501697	0.85281	0.8566	0.86588	0.85688	0.85645
model_229	0.86031	0.83676	0.67499	0.78724	0.87235	0.80131	0.534827	0.83754	0.82935	0.87235	0.84563	0.83683
model_96	0.8591	0.85869	0.6656	0.80298	0.86862	0.84906	0.498623	0.85352	0.8468	0.86862	0.85047	0.85884
model_235	0.85827	0.77243	0.5424	0.7248	0.90938	0.63708	0.590595	0.77752	0.74201	0.90938	0.80141	0.77323
model_94	0.85718	0.86759	0.68685	0.81611	0.86244	0.87307	0.486426	0.8608	0.86816	0.86244	0.85599	0.86776
model_211	0.85713	0.83998	0.67075	0.79545	0.86696	0.8131	0.53592	0.83754	0.83247	0.86696	0.8454	0.84003
model_220	0.8568	0.85633	0.69728	0.81332	0.86184	0.85097	0.515502	0.85103	0.86225	0.86184	0.85409	0.8564

Table 15: Results - MAML with FP, using top 20 columns - aggregated by f2 score

With FP - Top 20 - BSS score

model	f2	gmean	bss	pr_auc	sensitivity	specificity	ppv	accuracy	precision	recall	f1	auc
model_35	0.88339	0.88793	0.76578	0.85218	0.88299	0.89292	0.505276	0.88406	0.89551	0.88299	0.88596	0.88796
model_36	0.86405	0.87819	0.74672	0.84604	0.8589	0.89754	0.492165	0.87393	0.8953	0.8589	0.87369	0.87822
model_234	0.87664	0.87315	0.74385	0.83544	0.87891	0.8675	0.521648	0.87305	0.88027	0.87891	0.87557	0.87321
model_215	0.87711	0.8784	0.74115	0.83454	0.87953	0.87732	0.517793	0.87518	0.87699	0.87953	0.87527	0.87842
model_49	0.87003	0.87359	0.74059	0.83558	0.86997	0.87724	0.49026	0.87145	0.88043	0.86997	0.87204	0.87361
model_213	0.86712	0.87315	0.73672	0.83865	0.86648	0.87993	0.501388	0.86914	0.88384	0.86648	0.87059	0.8732
model_54	0.87615	0.87149	0.73155	0.82762	0.88201	0.86107	0.521293	0.86701	0.86871	0.88201	0.87037	0.87154
model_42	0.85256	0.87068	0.73032	0.83933	0.84585	0.89562	0.49065	0.86506	0.8955	0.84585	0.86556	0.87073
model_31	0.85406	0.8684	0.72885	0.83289	0.84841	0.88847	0.485661	0.86435	0.88763	0.84841	0.86447	0.86844
model_37	0.86253	0.86849	0.7251	0.82838	0.86191	0.87512	0.496969	0.86506	0.87567	0.86191	0.86537	0.86851
model_222	0.85549	0.86128	0.71728	0.82668	0.85473	0.86792	0.50531	0.85795	0.87689	0.85473	0.85999	0.86132
model_93	0.85665	0.86942	0.71603	0.82083	0.85574	0.88315	0.471064	0.86665	0.87268	0.85574	0.8603	0.86944
model_144	0.84199	0.86218	0.71158	0.82754	0.83535	0.88908	0.498168	0.85831	0.88559	0.83535	0.85505	0.86221
model_216	0.83795	0.85744	0.70727	0.82415	0.83095	0.88407	0.488089	0.85352	0.88507	0.83095	0.85167	0.85751
model_394	0.8335	0.8581	0.70534	0.8276	0.82502	0.89134	0.480615	0.85494	0.88508	0.82502	0.84926	0.85818
model_219	0.86477	0.85635	0.70387	0.81352	0.87298	0.83992	0.520798	0.85121	0.85694	0.87298	0.85716	0.85645
model_41	0.86115	0.85642	0.7005	0.81025	0.86588	0.84702	0.501697	0.85281	0.8566	0.86588	0.85688	0.85645
model_220	0.8568	0.85633	0.69728	0.81332	0.86184	0.85097	0.515502	0.85103	0.86225	0.86184	0.85409	0.8564
model_40	0.84086	0.8532	0.69509	0.8133	0.83752	0.86902	0.492097	0.84925	0.87203	0.83752	0.84902	0.85327
model_127	0.84588	0.856	0.69501	0.81399	0.84613	0.86602	0.498951	0.8489	0.86499	0.84613	0.84908	0.85608

Table 16: Results - MAML without FP, using top 20 columns - aggregated by bss score

21.7 Insights

1. We realize that the highest F2 score reached was by an MAML model, that **incorporated** FP Growth, and taking the top 10 columns selected by feature selection. **The highest F2 score is 0.89 shown in Table 13**

2. We realize that the highest BSS score reached by an MAML model, that **incorporated** FP Growth, and taking the top 20 columns selected by feature selection. **The highest BSS score is 0.76 shown in Table 16**

3. the model named *'model_35'* found in Table 16 seems to be a perfect model because, although not the highest in f2 score, it has an F2 score of 0.88, a GMEAN score of 0.88, a PR_AUC score of 0.85, and a BSS score of 0.76. Therefore, this model is doing equivalently well in performance with regards to predicting the positive and the negative class.

21.8 Insights on With FP vs. Without FP

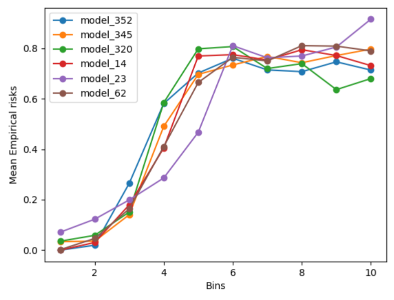
In terms of error metrics, the differences are marginal between the experiments incorporating and those not incorporating FP growth. This observation is expected because, in both cases, we have modified the randomness in generating tasks for training the MAML model, in such a way that in both we made sure to generate tasks from all the heterogeneous data samples found in the dataset. However, although this is just a *'heuristic'*, but it turns out to capture enough of the data. **This heuristic covers tasks from all around the dataset in a *round robin fashion***

21.9 Insights about PPV

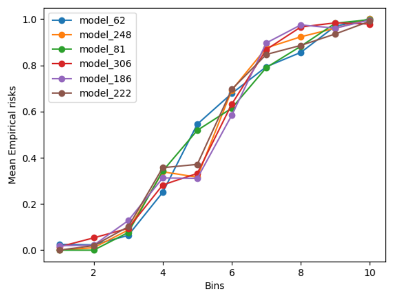
Its is important to note that in most of the results, we realize that PPV (**P**robability **P**ositi**V**e Class) was always threshold approximately around 0.6, expect in Table 13 were we see that the top 20 best performing models had PPVs of 0.9. This means that these models were able to assign high risks to some of the instances, which means that they were able to assign high risks to patients that are most likely to be demented. This effect will be captured in the **empirical risk curves**

21.10 Qualitative Results

21.11 Mean Empirical Risk Curves

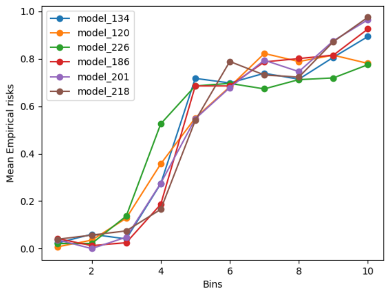


(a) Mean Empirical Risks for top 6 models - Without FP - top 10 columns

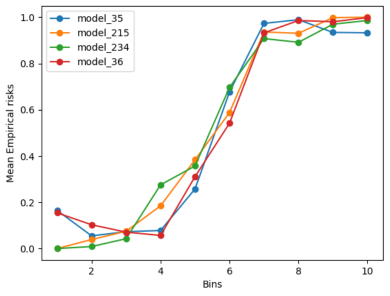


(b) Mean Empirical Risks for top 6 models - Without FP - top 20 columns

Figure 16: Mear Empirical Risks for top models - Without FP - trained on top 10 and top 20 columns selected by feature selection



(a) Mean Empirical Risks for top 6 models - With FP - top 10 columns



(b) Mean Empirical Risks for top 6 models - With FP - top 20 columns

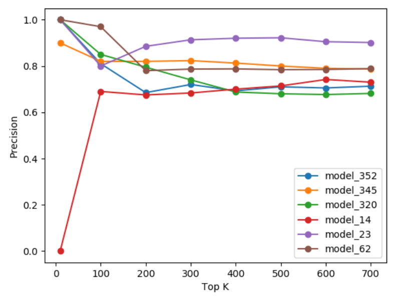
Figure 17: Mean Empirical Risks for top models - With and Without FP - trained on top 10 and top 20 columns selected by feature selection

In order to produce empirical risk curves, first, we rank patients by descending order of their estimated risk scores. We then group patients into bins based on the percentiles they fall into when categorized using risk scores. In our experiments, we choose to create 10 bins. The bottom 10% of patients who have the least risk are grouped into a single bin. Those who rank between 10th and 20th percentile are grouped in the next bin and so on. For each such bin, we compute the *empirical risk score* **which is the fraction of patients from that bin who actually, as per ground truth, are demented**. A good model would be classifying patients correctly if the *empirical risk curve* is monotonically non-decreasing.

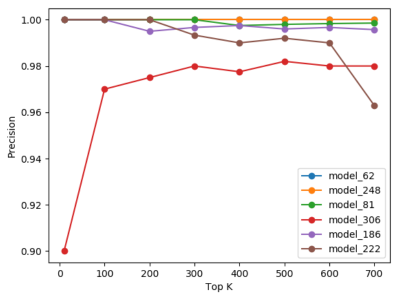
If the empirical risk curve is non-monotonic for some models, it implies that the classification using the model's risk scores may result in scenarios where patients with lower risk scores are more likely to be demented compared to patients with higher risk scores.

In the plots above, we realize that, whether with FP or without FP, the models that are trained using top 20 features have better empirical risk curves than those trained using top 10 features.

21.12 Precision at Top K

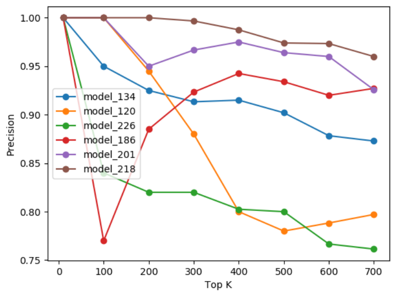


(a) Precisions for top 6 models - Without FP - top 10 columns

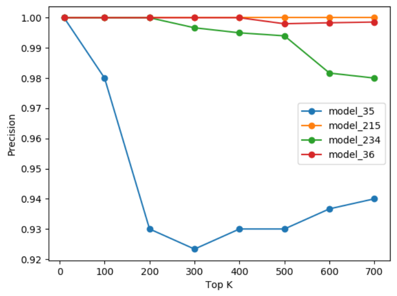


(b) Precisions at Top K for top 6 models - Without FP - top 20 columns

Figure 18: Precisions at Top K for top models - Without FP - trained on top 10 and top 20 columns selected by feature selection



(a) Precisions at Top K for top 6 models - With FP - top 10 columns



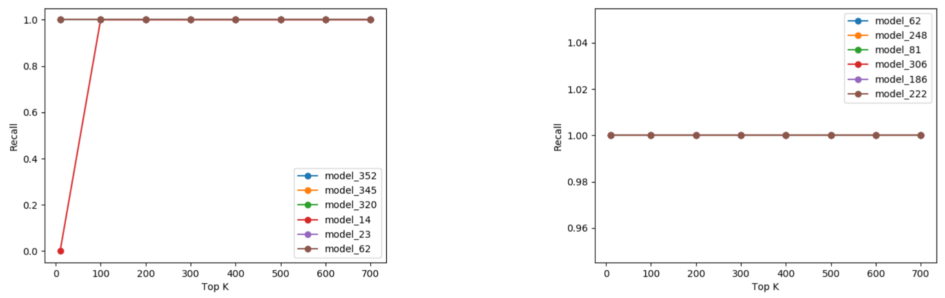
(b) Precisions at Top K for top 6 models - With FP - top 20 columns

Figure 19: Precisions at Top K for top models - With and Without FP - trained on top 10 and top 20 columns selected by feature selection

It might happen that hospitals, for example, might want to admit only a certain number of patients, and for that, it might want to admit only the patients that are at a very high risk of having dementia. For that reason, clinicians might be interested in models that provide good risk estimates to rank . Therefore, it might be very helpful to provide the precision/recall values of various models at different values of K.

It is good to note that in all 4 figures above, the precision values do not drop a lot for smaller K values (with the exception of model_14 and model_306 in the first two upper figures). Realize that in the bottom figures, the range of values for precision is still between [0.7-1] and [0.92-1]

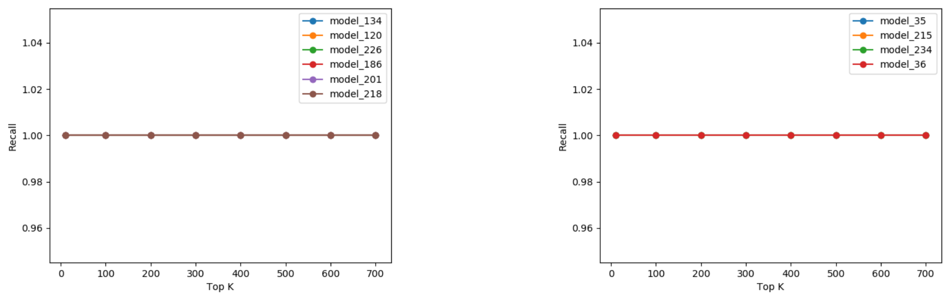
21.13 Recall at Top K



(a) Recalls for top 6 models - Without FP - top 10 columns

(b) Recalls at Top K for top 6 models - Without FP - top 20 columns

Figure 20: Recalls at Top K for top models - Without FP - trained on top 10 and top 20 columns selected by feature selection



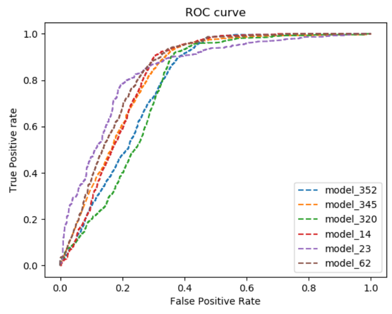
(a) Recalls at Top K for top 6 models - With FP - top 10 columns

(b) Recalls at Top K for top 6 models - With FP - top 20 columns

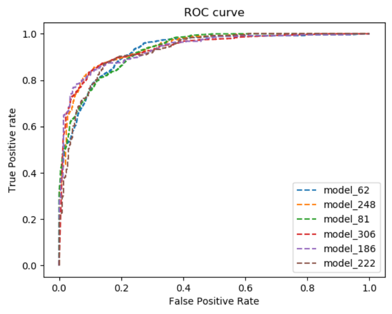
Figure 21: Recalls at Top K for top models - With and Without FP - trained on top 10 and top 20 columns selected by feature selection

We realize that the recall values at top K stay stable at 1 with the exception of the model_14 in the first figure; However, model_14 increases back to reach 1.

21.14 ROC Curves

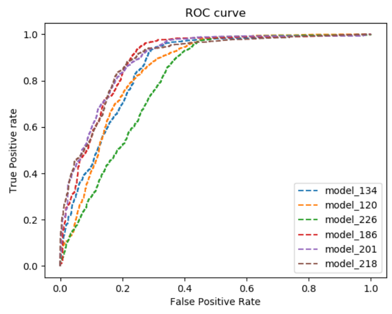


(a) ROC Curves - Without FP - top 10 columns

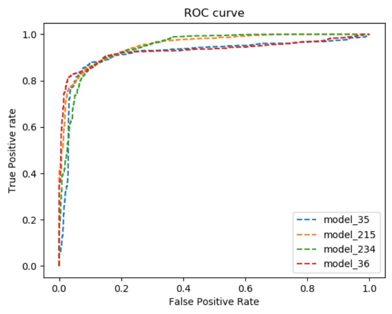


(b) ROC Curves for top 6 models - Without FP - top 20 columns

Figure 22: ROC Curves for top models - Without FP - trained on top 10 and top 20 columns selected by feature selection



(a) ROC Curves for top 6 models - With FP - top 10 columns



(b) ROC Curves for top 6 models - With FP - top 20 columns

Figure 23: ROC Curves for top models - With and Without FP - trained on top 10 and top 20 columns selected by feature selection

We realize that the best ROC Curves are those of the models trained on the top 20 features selected by feature selection, whether with or without FP.