Winning Model Documentation

Name: Matt Berseth

Location: Jacksonville FL

Competition: Practice Fusion Diabetes Classification

1. Summary

To extract features I created a script that allowed me to create features by specifying individual diagnosis codes, groups of diagnosis codes or a fuzzy match on the diagnosis description. I have provided more details on this in section 2.

I used boosted trees as my base learners. I blended my best learners linearly and used the coefficients to weight the contribution of each individual learner.

The primary data source that I used was wikipedia's page on type 2 diabetes [1]. I went through this page and mapped each of the complicating conditions with their ICD9 codes / groups. In addition to wikipedia, I also leveraged HCUP [2] and cross-walked some of the ICD9 codes to their ICD10 groupings [3].

2. Features Selection / Extraction

In total, I extracted nearly 400 features from the training data. These features can be categorized into the following buckets:

1. Patient features: {Age, Gender, BMI, blood pressure, etc ...}
    - These features were derived from the transcript table. Here is a sample of these features:
        - Age
        - BodyMassIndex
        - Weight
        - MaxWeight
        - MinWeight
        - DiastolicBP
        - SystolicBP
        - etc ...
2. Diagnosis features: {Most frequent ICD9 code, Has occurrence of specific code like hyperlipidemia or hypertension or any of the other ICD9 codes that could be considered complicating conditions.}
    - These features were derived from the diagnosis tables. Here is a sample of these features:
        - ICD9_01, ICD9_02 ... ICD9_07
        - HasAlzheimers
        - HadStroke
        - HasMixedHyperlipidemia
        - HasRenalDisease
    - I also found it helpful to include counts for diagnosis codes that were being treated with medications. Here is a sample of these features:
        - MedDxHasCongestiveHeartFailure
        - MedDxHasDepression
        - MedDxHasHyperlipidemia
        - MedDxHasRenalDisease
        - MedICD9_01 ... MedICD9_04
    - Finally, I also found it useful to look at the ICD9 level groupings. So I included features for each of the levels:
        - DxLevel1_280_289
        - DxLevel2_240_246
        - DxLevel2_249_259
        - etc ...
3. Medication features: {Most frequent NDC/drug name, Has occurrence of specific NDC/drug name}
    - These features were derived from the medication tables. Here is a sample of some of these features:
        - NDC_01 .. NDC_04
        - IsOnCoregOralTablet
        - IsOnCozaarOralTablet
        - IsOnLisinopril
        - etc ...
4. Provider features: {Specialty of the providers that were visited, Did they visit providers the other diabetics visited}
    - These features were derived from the transcript table as well as the diagnosis and medication tables:
        - SawDiabetesSpecialist
        - SawNephrologySpecialist
        - SawPodiatrySpecialist
        - SawPrescriptionUser_1353
        - SawTranscriptUser_104

    - etc ...
5. Counts: {Number of Diagnosis, Number of Medications, Number of Transcripts, etc ...}
    - These features are aggregate counts from all of the tables:
        - DXUniqueCount
        - HL7TextUniqueCount
        - MedUniqueCount
        - MedPerTranscriptCount
        - MedPerTranscriptYears
        - DxRelatedToDiabetesCount
        - PrescriptionCount
        - SpecialtyCount
        - etc ...
6. Higher Order Features
    - I also clustered the patients based on different groups of the above features and added features for the probability a given patient was in each of the clusters. I used the same clusters to add features marking each patient with an outlier probability based on the cluster model:
        - DMX_DrugProfile_ClusterNumber
        - DMX_DrugProfile_Cluster1Prob ... DMX_DrugProfile_Cluster5Prob
        - DMX_DxLevelClusters_ClusterNumber
        - DMX_DxLevelClusters_Cluster1Prob ... DMX_DxLevelClusters_Cluster5Prob
    - I also included a set of features for comorbidities. These features were counts of the number of major chronic conditions a patient was dealing with:
        - CoMorbidityCount
        - HasHypertensionAndHyperlipidemia
        - OnlyCoMorbidityIsHeartDisease
7. Data Features
    - Features that are a by product of how the data was collected
        - NullTranscriptDataCellPercent
        - MedNameIsNull
        - MedHasNoCorrespondingDx
        - etc ...

From these feature groups, I selected feature by doing the following:

1. Looked at [1] to determine what conditions commonly occur with type 2 diabetes
2. Looked at the data and determined what features listed above commonly occur with the patients that are marked as having type 2 diabetes
3. Used random forests and boosted trees to rank and weight the features
4. Used a genetic algorithm to search the feature space

---

3. Modeling Techniques and Training

My base learners were boosted trees. I used cross validation on the training data to select model parameters to use. My best submissions used some where between 275 to 375 trees, a learning rate of 0.1, had a maximum depth of 3 and required 20-30 cases in each of the leaf nodes.

I had different learners based on different sets of the above features. These base learners were blended linearly using the weights of the coefficients to weight each of the learners contribution to the overall prediction.

I also used clustering models to segment the patients to develop a drug profile, patient profile (age, weight, bmi, etc ...) and a diagnosis profile. These models were also used to generate outlier features (i.e. given the cluster model, how likely is the case data).

The external data sources I used was primarily wikipedia [1]. I also included features from HCUP [2] and ICD10 cross-walk [3].

---

4. Code Description

I used a combination of .Net/C#, SQL Sever and Python to create my solution. I stored the data in a sql server database and used a utility console application to generate the features from the training tables. After the features were generated, I exported the training and test case tables to flat files and used this as input into my python scripts that contained the actual model.

The source tree I used has the following structure

```
/source
    /attic
    /extternal_data
    /last_weekend_notes
    /net
    /notes
    /python
    /R
    /sql
```

The `attic` is for any prototype code. `external_data` was to keep track of my data sources. `last_weekend_notes` was from my last push at finalizing my model. I had started to look at the gbm R package and compare that to the python packages I was using. `R` contains some of this work as well. `notes` has general notes that I was taking along the way, mostly regarding the feature selection.

The `sql` `net` and `python` folders are the most important directories. `sql` contains a number of scripts that I used to clean the data. It also contains the analysis services project I used to create the cluster models.

The `net` folder contains a visual studio project and corresponding C# code for generating the features. The solution, `ml_practice_fusion`, contains 3 projects:

```
ml_practice_fusion
ml_practice_fusion.etl
ml_practice_fusion.etl.features
```

`ml_practice_fusion` contains common code like connection strings and data access logic. `ml_practice_fusion.etl` is the driver application and entry point. It also contains some basic file parsing and generation logic.

Finally, the majority of code in the last project, `ml_practice_fusion.etl.features`. This project has a class file for each feature. The class is responsible for adding the column to the test and training case tables and populating it with the proper value.

For example, some of my most important features were groups of ICD9 codes - either matching a pattern on the actual code, or the diagnosis description. For these types of features, I created a base class that would handle the generic processing and then have the derived class specify the actual expression that needs to be matched on.

Concretely, here is the class definition for my `HasHypertension` feature:

```
public class DxFeature_HasHypertension : DxFeature
{
    public DxFeature_HasHypertension()
    : base()
    {
        this.ColumnName = "HasHypertension";
        this.LikeExpressions.Add("40[1-5]%");
    }
}
```

You can see, this class is just metadata, defining the column name and the like expression to use. A majority of the features are generated following a similar pattern.

All of the models are created using the python scikit-learn [4] package. Like the .net project, I have a couple of files for common things. The `io.py` file contains logic to read/write csv files. `score.py` implements the log loss function and `features.py` contains different listing of features.

Individual models, the base learners I used as input into my blending/stacking routine, were trained using `model.py`. `model.py` contains an eval function that is passed a set of features and a tree classifier from the scikits ensemble module. I used this common function to train and cross validate `GradientBoostingClassifier`, `ExtraTreesClassifier` and `RandomForestClassifier`. The `run.py` file was used to setup the model and features and then call the model.eval function. Here is an example:

```
import numpy as np
import model
import features
from sklearn.ensemble import GradientBoostingClassifier as gb

f = features.submission21
loss = model.eval(f, gb(n_estimators=300, learn_rate=.1, max_depth=3, random_state=1))
print 'Avg Loss:', np.mean(loss), np.std(loss)
```

In this example, I am using the features I used for my 21st submission, passing this and the `GradientBoostingClassifier` to my `model.eval` function. `model.eval` returns the log loss for each fold of the cross validation.

Blended models are trained and cross validated using the `stack.py`. Like `model.eval`, `stack` has a similar interface, except it takes a list of features and models. Here is an example:

```
items = [
    {
        'clf': gb(n_estimators=400, learn_rate=.09, max_depth=3, max_features=10, min_samples_leaf=30, random_state=1),
        'features': features.submission21
    },
    {
        'clf': gb(n_estimators=400, max_depth=3, max_features=10, min_samples_leaf=30, learn_rate=.09, random_state=1),
        'features': features.next_submission_v2
    },
    {
        'clf': gb(n_estimators=400, max_depth=3, max_features=10, min_samples_leaf=30, learn_rate=.09, random_state=1),
        'features': features.beat_blind_ape
    }
]
eval(items)
```

`grid_search.py` is used to attempt tune the model parameters. The top of this file includes custom classifiers that allow plugging in the log loss for the evaluation/scoring. I used `ga.py` to search the feature space to attempt to identify groups of features that worked well together.

`submit.py` contains the logic for submitting a blended model.

5. How to Generation the Solution (aka README)

The training and test data are in the root of the solution directory. They are `dbo.training_Features.csv` and `dbo.test_Features.csv`. To recreate the solution, do the following:

I used the following 5 submissions as my final submissions

1. submission 41

```
items = [
    {
        'clf': gb(n_estimators=300, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.next_submission_v2
    },
    {
        'clf': gb(n_estimators=275, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.next_submission_v1
    },
    {
        'clf': gb(n_estimators=300, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.next_submission_v1
    },
    {
        'clf': gb(n_estimators=275, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.submission21
    },
    {
        'clf': gb(n_estimators=325, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.submission21
    },
    {
        'clf': gb(n_estimators=375, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.submission21
    },
    {
        'clf': gb(n_estimators=275, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.submission21
    },
    {
        'clf': gb(n_estimators=325, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.submission21
    },
    {
        'clf': gb(n_estimators=375, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.submission21
    },
    {
        'clf': gb(n_estimators=400, learn_rate=.09, max_depth=3, max_features=10, min_samples_leaf=30, random_state=1),
        'features': features.submission21
    },
    {
        'clf': gb(n_estimators=400, max_depth=3, max_features=10, min_samples_leaf=30, learn_rate=.09, random_state=1),
        'features': features.next_submission_v2
    },
    {
        'clf': gb(n_estimators=400, max_depth=3, max_features=10, min_samples_leaf=30, learn_rate=.09, random_state=1),
        'features': features.beat_blind_ape
    },
    {
        'clf': gb(n_estimators=400, max_depth=3, max_features=10, min_samples_leaf=30, learn_rate=.09, random_state=1),
        'features': features.next_submission_v1
    },
]
```

2. submission 38

```
items = [
        {
        'clf': gb(n_estimators=300, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.next_submission_v2
    },
        {
        'clf': gb(n_estimators=275, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.next_submission_v1
    },
        {
        'clf': gb(n_estimators=300, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.next_submission_v1
    },
        {
        'clf': gb(n_estimators=275, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.submission21
    },
        {
        'clf': gb(n_estimators=325, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
        'features': features.submission21
    },
        {
        'clf': gb(n_estimators=375, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
            {
            'clf': gb(n_estimators=275, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
            {
            'clf': gb(n_estimators=325, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
            {
            'clf': gb(n_estimators=375, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
            {
            'clf': gb(n_estimators=300, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.beat_blind_ape
        },
            {
            'clf': gb(n_estimators=300, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.beat_blind_ape
        }
    ]
```

3. submission 36 / 37

```
    items = [
            {
            'clf': gb(n_estimators=300, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.next_submission_v2
        },
            {
            'clf': gb(n_estimators=275, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.next_submission_v1
        },
            {
            'clf': gb(n_estimators=300, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.next_submission_v1
        },
            {
            'clf': gb(n_estimators=275, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
            {
            'clf': gb(n_estimators=325, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
            {
            'clf': gb(n_estimators=375, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
            {
            'clf': gb(n_estimators=275, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
            {
            'clf': gb(n_estimators=325, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
            {
            'clf': gb(n_estimators=375, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        }
    ]
```

4. submission 34

```
    items = [
            {
            'clf': gb(n_estimators=275, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
            {
            'clf': gb(n_estimators=325, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
            {
            'clf': gb(n_estimators=375, max_depth=3, min_samples_leaf=20, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
        {
            'clf': gb(n_estimators=275, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
        {
            'clf': gb(n_estimators=325, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        },
        {
            'clf': gb(n_estimators=375, max_depth=3, min_samples_leaf=30, min_samples_split=10, learn_rate=.1, random_state=1),
            'features': features.submission21
        }
    ]
```

To generate the above five submissions do the following:

1. Open `submit.py`
2. Check the value of the path variables, point those to the proper locations
3. Copy the `items` list from the submission above.
    - Note: submission 37 was trained on `dbo.training_Features_no_outliers.csv`, so you will need to update the training file path before you generate the scored file

6. Additional Comments and Observations

Trust your cross validation scores and use the public leaderboard as a measurement of competitiveness. When I selected my final models for submission, I picked the models with the lowest cross-validation scores, not the ones with the lowest public leaderboard scores. This worked well for me in this competition, using this formula, I ended up picking my five best models.

7. Figures

I did not generate any interesting figures or plots as part of my analysis.

8. References

1. `http://en.wikipedia.org/wiki/Diabetes_mellitus_type_2`
2. `http://www.ahrq.gov/data/hcup/`
3. `http://www.ama-assn.org/ama1/pub/upload/mm/399/crosswalking-between-icd-9-and-icd-10.pdf`
4. `http://scikit-learn.org/stable/`