modelling-evaluation-ammar-150454388

January 25, 2019

Table of Contents

Modelling and Evaluation Report

Ammar Hasan 150454388

In [1]:
```
# Imports

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
import sklearn.metrics as met
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier
import time


BASE_PROCESSED_DATA_DIR = '../data/processed'
"""
str: Base processed data directory
"""

PROCESSED_CSV_FILE = BASE_PROCESSED_DATA_DIR + '/processed.csv'
"""
str: HAM1000_metadata.csv metadata file location
"""

# Read dataset in
skin_df = pd.read_csv(PROCESSED_CSV_FILE, index_col=0)
"""
pandas.core.frame.DataFrame: final dataset
"""
def tuning(function, parameters, X_train, y_train):
    """
    Used to tune a model using 5 cv
    adapted from https://github.com/yuguan1/example-ML-code

    returns best parameters from GridSearchCV
    """
    print("# Tuning hyper-parameters")
    clf = GridSearchCV(function, parameters, cv=5)
    clf.fit(X_train, y_train)

    print('best parameters:')
    print(clf.best_params_)
    print('-------------------------------------')
    return(clf.best_params_)

def printMetrics(prediction, y_test):
```

```
            """
            Prints accuracy, confusion and F1 metrics

            returns list of accuracy, confusion and F1 metrics
            """
            accuracy = met.accuracy_score(y_test, prediction)
            confusion = met.confusion_matrix(y_test, prediction)
            f1_score_avg = met.f1_score(y_test, prediction, average='weighted')
            f1_score = met.f1_score(y_test, prediction, average= None)

            print('accuracy', accuracy)
            print()
            print(confusion)
            print()
            print('f1 average: ', f1_score_avg)
            print('f1: ', f1_score)

            return([accuracy, confusion, f1_score_avg])

        lesion_type_label = skin_df[
            ['lesion_type_idx', 'lesion_type']].sort_values(
            'lesion_type_idx').drop_duplicates()['lesion_type']
        """
        pandas.core.series.Series: Lesion types (text) series sorted by idx for labels
        """
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWa
  from numpy.core.umath_tests import inner1d
```

```
Out[1]: '\npandas.core.series.Series: Lesion types (text) series sorted by idx for labels\n'
```

0.1 Report Introduction

This report documents the Modelling and Evaluation stages of the CRISP-DM process followed by this project. These stages documents the construction of models and their evaluation - and to a lesser extent the project itself. This includes background information about the models, the training of them and their evaluation.

0.2 Background

As stated in the Business Understanding report the Pigment Skin Diagnosis field use of technology has been growing due critical importance of early detection, and the computerised detection of Skin Lesions is becoming critical. In this project as stated in the criteria of Business Understanding report, the objective of this project is to develop, train and evaluate models (logistic, SVM and Random Forest) to classify skin lesion types and to compare the different models.

0.3 Modeling Assumptions and Potential Issues

As stated in the Data Understanding report, some issues with the data were recognised:

- Some of the sexes are unknown and the mention of whether these unknown sexes are simply unknown or are non-binary/non-conforming is missing.
- Ground truths were obtained by various methods including expert consensus, which might effect the consistency of the classification.
- The skin lesion condition frequencies are dominated by nevi and some conditions have very small frequencies in comparison (e.g. vascular and dermatofibroma)

Hence, we need to assume that the unknown sexes and the different methods to obtain the ground truth will not have a major impact on the modelling and its analysis, but as stated in the Data Understanding report these issues are not major. But, the skin lesion frequencies might cause some issues with some models.

0.4 Test design

As alluded to before in the background, the following classification models will be built in this project:

- A neural network - specifically a Convoluted Neural Network
- Logistic Regression Model
- A Support Vector Machine

Models are fitted by using a training data set and if tuning is required 5 fold cross validation is also used to find the optimal parameters. To test models the testing data set is used after the model was trained to evaluate the model (accuracy, confusion, f1 score, etc). The Training and test data is created by splitting the post processed final dataset described in the Data Understanding report in a 50-50 split respectively.

0.4.1 Creating the Testing and Training Datasets

The final dataset whose creation was described in the Data Preparation report needs minor changes before the modelling begins. In particular, the categorical variables need to be hot coded and the data needs to be divided into model training and model testing samples.

One Hot Encoding and Minor Manipulation

```
In [2]: # encode categorical cols using one hot encoding

        one_hot_localization = pd.get_dummies(skin_df['localization'])
        one_hot_localization.drop('unknown', axis=1, inplace = True)

        one_hot_sex = pd.get_dummies(skin_df['sex'])
        one_hot_sex.drop('unknown', axis=1, inplace = True)
```

```
        # Drop old categorical cols and replace with new ones
        # drop dx type (not needed beyond data understanding)

        skin_df.drop(['dx_type', 'localization', 'sex'], axis = 1, inplace = True)

        # Join the encoded dfs

        skin_df = skin_df.join(one_hot_localization)
        skin_df = skin_df.join(one_hot_sex)
```

Using pandas dummies for categorical variables, localization values are one hot coded using new columns for every value (0 false / 1 true), however one of the columns is dropped since a negation of all the other columns represents it. Lastly, the now redundant sex and localization fields are dropped alongside dx_type (no need for analysing diagnosis type beyond Data Understanding).

Test Split and Scaling

```
In [3]: # Split the dataset into training and test data in a 50-50 split
        # Don't include lesion_types (used for response) and image path (not used yet)

        X_train, X_test, y_train, y_test = train_test_split(
            skin_df.drop(['lesion_type_idx', 'lesion_type'], axis=1),
            skin_df['lesion_type_idx'], test_size=0.5, random_state=0)

        # scale using a partial fit for speed

        scaling = StandardScaler()

        scaling.partial_fit(X_test)
        X_test = scaling.transform(X_test)

        scaling.partial_fit(X_train)
        X_train = scaling.transform(X_train)
```

The training data and the testing data are separated using a 50-50 split respectively, both sets consist of a set of predictors (X) and a response (y). The predictor data has the lesion_type_idx and lesion_type fields removed since they can leak the ground truth. For the response data only the lesion_type_idx field is used since it is sufficient at representing the category of skin lesion (the response / what is being predicted).

To ensure that the impact of predictors is not effected by the measurement scale - which could occur in this dataset due to the variety of predictors - the predictors are scaled using a scaling transform (i.e. with default mean and standard deviation transform).

0.4.2 Measurements

The computer used to carry the measurements has the following specifications: * CPU: i7-7700HQ * RAM: 8GB * OS: Windows 10 * GPU: GTX 1060 (notebook)

To evaluate the model the following measurements are taken: * Fit time: Using the time python library, a timer is started and stopped to measure tuning and fit . * Prediction time: Using the time python library, a timer is started and stopped to measure the prediction. * Confusion matrix: Using the sklearn metrics library a confusion matrix is printed. * F1 Score: Using the sklearn metrics library a F1 score is calculated using weighted averages and for every class.

0.5 Models Description and Assessments

0.5.1 Logistic Regression

Introduction To achieve its classification, Logistic Regression fits a line to separate the data into classes. The line is fitted by minimising the error between line and points by changing the coefficients/weights and the intercept to find the "ideal fit". To minimise the error gradient descent (i.e. loss function optimisation) is used, which updates the parameters (i.e. through partial derivatives) to find a local minimum/maximum. (James et al., 2013)

Construction and Tuning

In [4]:
```
# fit with time measurements, use n_jobs -1 for all cores

        log_fit_start = time.time()
        clf = LogisticRegression(solver='saga', n_jobs = -1).fit(X_train, y_train)
        log_fit_end = time.time()

        log_fit_time = log_fit_end - log_fit_start
        print('Logistic fit time(seconds): ', log_fit_time)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326: ConvergenceWarning:
  "the coef_ did not converge", ConvergenceWarning)


Logistic fit time(seconds):  28.128968238830566
```

As previously stated, logistic regression fits a line through error minimisation and does not require tuning as the objective is straightforward. The model is simply constructed using a saga solver deployed to maximum threads (-1) using sklearn's logistic regression model.

Nonetheless, the constructed model does not seem to converge, this is not that surprising as the fitted line can have many solutions, and sometimes with a large number of predictors this can cause logistic regression to struggle to find an ideal line (James et al., 2013).

Interpretation

In [5]:
```
# form table for coefficients using column names

        coefs = pd.concat([pd.DataFrame(skin_df.drop(['lesion_type_idx','lesion_type'], axis = 1
                           pd.DataFrame(np.transpose(clf.coef_), columns = lesion_type_label)],

        # print table

        print(coefs.iloc[3136:])
```

	0	Actinic keratoses	Basal cell carcinoma	\
3136	age	0.177438	0.344516	
3137	abdomen	-0.047619	-0.029246	
3138	acral	-0.020823	-0.016253	
3139	back	-0.097481	0.143565	
3140	chest	-0.025989	0.137862	
3141	ear	0.002960	-0.044507	
3142	face	0.248055	0.108345	
3143	foot	-0.068287	-0.081592	
3144	genital	-0.025291	-0.024418	
3145	hand	0.034297	-0.073172	
3146	lower extremity	0.060997	-0.092365	
3147	neck	0.014438	0.058447	
3148	scalp	0.077900	-0.007582	
3149	trunk	-0.112059	-0.036233	
3150	upper extremity	0.102047	-0.035125	
3151	female	-0.064417	0.014538	
3152	male	0.082081	0.013225	

	Benign keratosis-like lesions	Dermatofibroma	Melanocytic nevi	\
3136	0.577055	-0.051253	-1.046366	
3137	-0.058280	-0.005779	0.122986	
3138	-0.056595	-0.012838	0.083335	
3139	-0.010787	-0.059636	-0.022806	
3140	0.064507	-0.026130	-0.097631	
3141	-0.065038	-0.015696	-0.057262	
3142	0.315165	-0.049256	-0.623519	
3143	-0.228383	-0.009924	0.272416	
3144	0.014746	0.012924	0.044719	
3145	0.044948	-0.030387	0.041499	
3146	-0.023325	0.149217	-0.052326	
3147	0.089153	-0.014975	-0.143592	
3148	0.023331	-0.043749	-0.062815	
3149	0.047929	-0.041920	0.084017	
3150	-0.060397	0.074843	-0.127507	
3151	0.041676	-0.000378	-0.051918	
3152	-0.038489	0.014207	-0.065951	

	Melanoma	Vascular lesions
3136	0.420180	-0.041501
3137	-0.074304	0.055192
3138	-0.038075	-0.007010
3139	0.118811	-0.019974
3140	0.031109	-0.010167
3141	0.111783	-0.009440
3142	0.031731	0.012701
3143	0.022386	-0.046889
3144	-0.106234	-0.034480

```
3145 -0.073725          0.003845
3146  0.055989         -0.022239
3147  0.067730          0.019966
3148 -0.009795          0.010683
3149 -0.144810          0.077741
3150  0.186057         -0.016172
3151 -0.010818          0.032539
3152  0.079085         -0.014805
```

Since as described in the introduction of logistic regression, logistic regression uses a set of coefficients and weights to draw the classification line, it is easy to interpret the model by examining them. However, since there are many predictors we will only look at the meta data.

Here it can be seen that age is the strongest overall predictor, in particular with classes Basal cell carcinoma, Benign keratosis and Melanocytic nevi, where in carcinoma and keratosis age had a positive impact, but in nevi age had a strong negative one. Some of these classes were predicted to have a strong dependence on age in the Data Understanding stage (e.g. carcinoma). Age seems to mostly have a positive trend nonetheless - which matches with the observations in the data understanding stage.

When it comes to sex, three classes come to attention, Actinic keratoses, Melanocytic nevi and Melanoma. In keratoses, being a male has a noticiable positive impact which is also seen with class. In nevi females have a noticiable negative impact. However, in comparison to age, the sexes impact is minor.

For localization, the face seems to have a strong impact when it comes to Actinic keratoses. Also in Basal cell carcinoma, the face also has a strong impact on a classification in this class while the foot has the opposite effect. However, the classes mostly effected by localization are Melanocytic nevi where a localization in the face very strongly suggests against a classification to it. Overall, localization is the second strongest predictor here.

Assessment

```
In [6]: # carry prediction with time measurements
        # while recording prediction

        log_pred_start = time.time()
        prediction = clf.predict(X_test)
        log_pred_end = time.time()

        log_met = printMetrics(prediction, y_test)
        log_pred_time = log_pred_end - log_pred_start
        print('Logistic prediction time(seconds): ', log_pred_time)
```

```
accuracy 0.7330271565495208

[[  31   25   60    0   32    9    0]
 [  14  105   57    0   74    9    0]
 [   8   28  244    0  192   65    0]
 [   5   13    4    1   38    3    0]
```

```
 [   5   23   82    0 3153   93    0]
 [   7   14   97    0  310  132    1]
 [   0   14    5    0   46    4    5]]

f1 average:  0.7021641492370041
f1:  [0.27312775 0.43659044 0.44935543 0.03076923 0.87571171 0.30136986
 0.125     ]
Logistic prediction time(seconds):  0.03888416290283203
```

We can observe that while the overall accuracy and f1 score is good, the class F1 scores and poor except for the 5th class, this is worrying because as stated before in the business understanding false negative performance is important, and a poor F1 suggests a poor false negative performance as can be seen in the confusion table. However, the other classes have significantly less members and the f1 scores might not be accurate enough.

Nonetheless, this could be because of the issue observed in the model construction, as there was no convergence due to the large number of predictors. This might be fixed with the introduction of kernels as explained in the next section.

0.5.2 Support Vector Machines (SVM)

Introduction Support Vector Machines are another classification model, which is an improvement logistic regression through the addition of two margins for the line which all class points must be behind and the distance to them from the line maximised (again using gradient descent). This ensures that the fairest line out of a set of lines that separate the points is chosen. (Rieck et al., 2012)

To help interpreting non linear data for SVMs, kernels (commonly a Raidial Bias Function (RBF)) transform the data to a higher dimension (kernel trick) (Rieck et al., 2012).

Construction and Tuning

In [7]:
```
# Fit

        svm_fit_start = time.time()

        n_estimators = 10
        clf = OneVsRestClassifier(
            BaggingClassifier(
                SVC(kernel='rbf', random_state=0),
                max_samples=1.0 / n_estimators, n_jobs=-1,
                n_estimators=n_estimators, random_state=0
            ), n_jobs=-1
        ).fit(X_train, y_train)

        svm_fit_end = time.time()
        svm_fit_time = svm_fit_end - svm_fit_start
        print('SVM fit time(seconds): ', svm_fit_time)
```

```
SVM fit time(seconds):  24.3392550945282
```

Due to their high complexity (more on that later) a Bagging Classifier with a One Vs Rest Classifier (OvR) is used to help speed up the execution. The OvR basically creates a classifier for every class of a multi class classification problem to improve efficiency - enables paralysation - ("sklearn.multiclass.OneVsRestClassifier — scikit-learn 0.20.2 documentation," n.d.).

On the other hand, the Bagging Classifiers which are used by the OneVsRestClassifier works like a Random Decision tree, as it fits multiple classifiers on random subsets of the original dataset which it then aggregates for the final decision ("sklearn.ensemble.BaggingClassifier — scikit-learn 0.20.2 documentation," n.d.). This ensemble approach reduces the size of the dataset and enables paralysation.

However, this significantly complicates the function and hence tuning is not possible using a GridSearchCV for instance.

Interpretation Support Vector Machine are more black box like due to their use of kernels, and hence difficult to interpret when it comes to predictor influence - the transformation into higher dimensions makes it difficult to find coefficients.

Detailed assessment

```
In [8]: # Prediction

        svm_pred_start = time.time()
        prediction = clf.predict(X_test)
        svm_pred_end = time.time()

        svm_met = printMetrics(prediction, y_test)
        svm_pred_time = svm_pred_end - svm_pred_start
        print('SVM prediction time(seconds): ', svm_pred_time)
```

```
accuracy 0.6888977635782748

[[   1    3   17    0  134    2    0]
 [   0   10   19    0  230    0    0]
 [   0    0  116    0  421    0    0]
 [   1    2    7    0   54    0    0]
 [   0    0   37    0 3319    0    0]
 [   0    0   36    0  521    4    0]
 [   0    0    0    0   74    0    0]]

f1 average:  0.5866640093284952
f1:  [0.01257862 0.0729927  0.30169051 0.         0.81859662 0.01410935
 0.        ]
SVM prediction time(seconds):  155.82727026939392


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetr
  'precision', 'predicted', average, warn_for)
```

The accuracy seems to be down from the logistic regression, but more worryingly the f1 scores seem to have been hurt. However, the confusion table again suggests that the performance in largest class seem to have improved, which again suggests that the varying skin condition frequencies through the classes are making it difficult to assess the f1 score. Nonetheless, the gain is not massive considering the long fit and prediction times.

0.5.3 Random Forest

Introduction A random forest is a modelling technique that makes use of meta estimation through the use of multiple decision trees whose results are combined to form a final decision. The reliance on multiple decision trees helps resolve issues such as overfitting - reduces variance for bias. ("3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.2 documentation," n.d.)

The decision trees themselves are simple non-parametric supervised learning classifiers that make decisions throughout a structure based on entropy of values - high entropy branches and nodes are at higher levels. ("1.10. Decision Trees — scikit-learn 0.20.2 documentation," n.d.)

Construction and Tuning

```
In [9]: for_fit_start = time.time()

        parameters = [{'max_depth': [22,45],
                       'n_estimators': [225, 550]}]

        # Tune parameters using 5 fold cross validation
        best_params = tuning(RandomForestClassifier(n_jobs = -1, random_state=0), parameters, X_


        # fit

        optimalDepth = best_params['max_depth']
        optimalEstimators = best_params['n_estimators']
        clf = RandomForestClassifier(random_state=0, max_depth = optimalDepth, n_estimators = op
        clf.fit(X_train, y_train)

        for_fit_end = time.time()
        for_fit_time = for_fit_end - for_fit_start
        print('Random Forest fit time(seconds): ', for_fit_time)

# Tuning hyper-parameters
best parameters:
{'max_depth': 45, 'n_estimators': 225}
-------------------------------------
Random Forest fit time(seconds):  261.1578071117401
```

The Random Forest Classifier is tuned using a GridSearchCV (cross validation of 5 folds) on two parameters, 'max_depth' and 'n_estimators'. 'max_depth' controls how deep the tree can be

constructed - reducing overfitting and simplifies model - and 'n_estimators' controls the number of trees that are created - cuts overfitting for underfitting.

Interpretation

```
In [10]: importances = pd.concat(
             [pd.DataFrame
              (skin_df.drop(['lesion_type_idx', 'lesion_type'], axis = 1).columns, columns = ['f
              pd.DataFrame(np.transpose(clf.feature_importances_), columns = ['importance'])], a

         print(importances.iloc[3136:])
```

```
                field  importance
3136              age    0.012616
3137          abdomen    0.000040
3138            acral    0.000000
3139             back    0.000102
3140            chest    0.000043
3141              ear    0.000018
3142             face    0.003687
3143             foot    0.000056
3144          genital    0.000000
3145             hand    0.000057
3146  lower extremity    0.000067
3147             neck    0.000086
3148            scalp    0.000078
3149            trunk    0.000060
3150  upper extremity    0.000117
3151           female    0.000056
3152             male    0.000049
```

This method is still difficult to interpret since there are many trees to explore. However, sklearn provides importances that are calculated through weighted information gain found through permutation (pjh2011, 2018). These feature importances revel that age is of very strong importance relative to other fields. But, interestingly the face has a strong importance as well, which could be because of Actinic keratoses and Basal cell carcinoma, which it effected in the Logistic Regression interpretation.

Assessment

```
In [11]: # Prediction and Assesmenmt

         for_pred_start = time.time()
         prediction = clf.predict(X_test)
         for_pred_end = time.time()

         for_met = printMetrics(prediction, y_test)
```

```
        for_pred_time = for_pred_end - for_pred_start
        print('Random Forest prediction time (seconds): ', for_pred_time)
```

```
accuracy 0.7240415335463258

[[  21   29   46    0   60    1    0]
 [   6   78   45    0  126    4    0]
 [   4   20  202    0  294   17    0]
 [   1   13   13    0   37    0    0]
 [   2   11   66    0 3253   24    0]
 [   2    5   76    0  406   72    0]
 [   0    2    1    0   70    1    0]]

f1 average:  0.6673408721740317
f1:  [0.21761658 0.37410072 0.40973631 0.         0.85582741 0.21176471
 0.        ]
Random Forest prediction time (seconds):  0.4009273052215576


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetr
  'precision', 'predicted', average, warn_for)
```

Again, it seems that while the f1 score and accuracy have improved compared to the support vector machine, the f1 scores are still low when it comes to some classes. This again could be because of the lack of samples for other skin lesion classes.

The run time of this model fit seems to take a very significant amount of time, this is probably because it is the only model with tuning, but even with a small set of parameters the cross verification seems to take too long, but this is probably due to the size of the dataset.

0.6 Summary of Modelling

```
In [12]: model_labels = ['Logistic', 'SVM', 'Rnd Forest']
         """
         List: model name labels used for plotting
         """
```

```
Out[12]: '\nList: model name labels used for plotting\n'
```

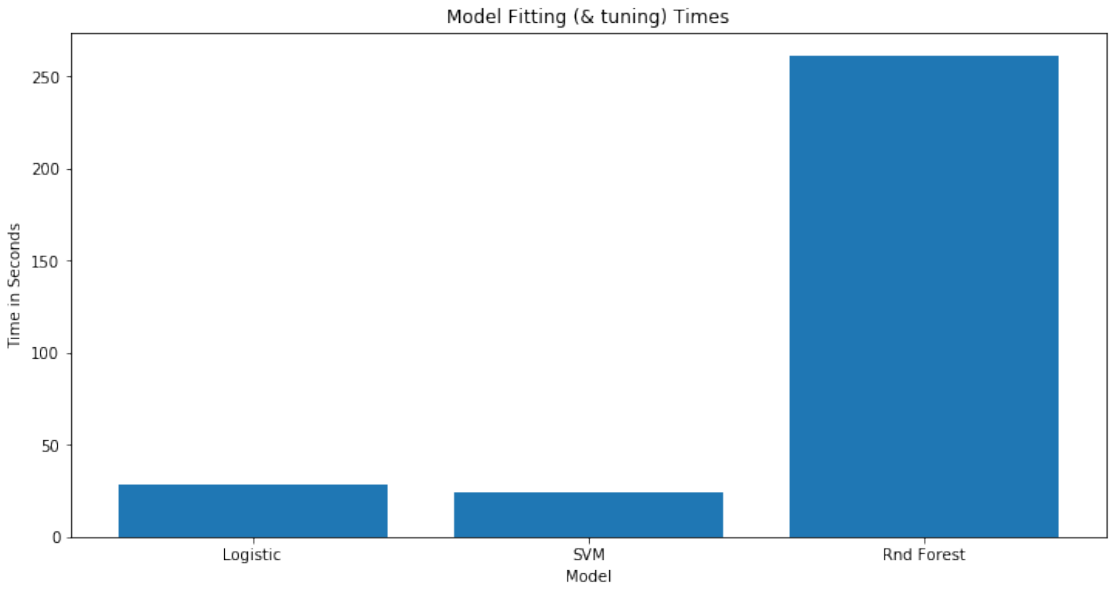
0.6.1 Fitting Times

```
In [21]: plt.bar([i for i, _ in enumerate(model_labels)],
                 [log_fit_time, svm_fit_time, for_fit_time])
         plt.title('Model Fitting (& tuning) Times')
         plt.ylabel('Time in Seconds')
         plt.xlabel('Model')
         plt.xticks([i for i, _ in enumerate(model_labels)],
                    model_labels)
```

```
plt.rcParams['figure.figsize'] = [12, 6]

plt.savefig('../reports/figures/fit-times.png') # save figure

plt.show()
```

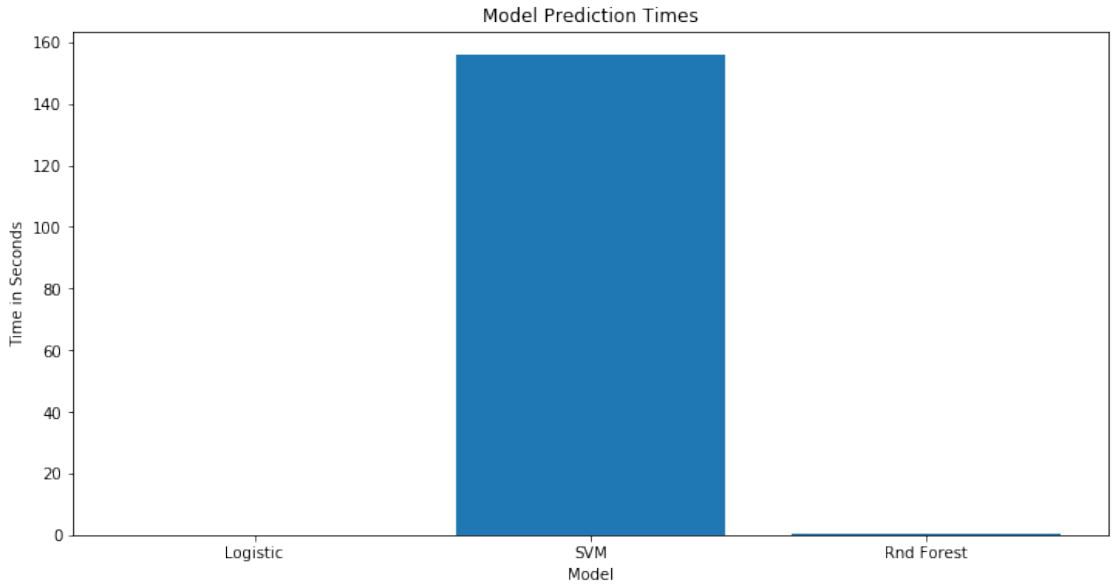


The Random Forest is overwhelmingly the slowest model to fit, being around 10 times slower.

0.6.2 Prediction Times

```
In [14]: plt.bar([i for i, _ in enumerate(model_labels)],
                 [log_pred_time, svm_pred_time, for_pred_time])
         plt.title('Model Prediction Times')
         plt.ylabel('Time in Seconds')
         plt.xlabel('Model')
         plt.xticks([i for i, _ in enumerate(model_labels)],
                    model_labels)

         plt.rcParams['figure.figsize'] = [12, 6]
         plt.show()
```

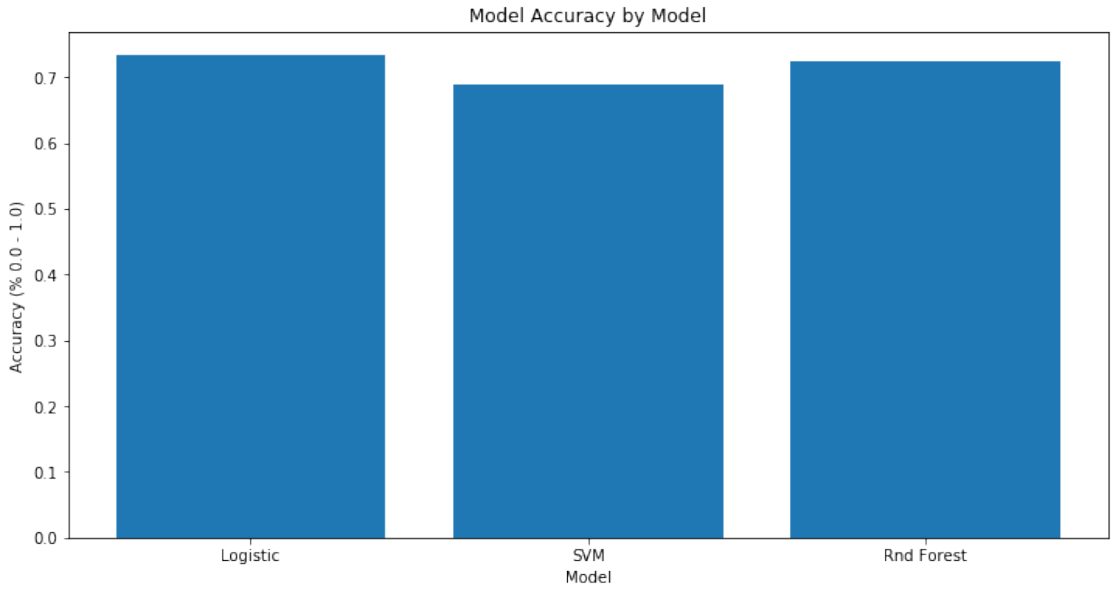


The SVM model is very tedious when it comes to predictions, being more than 100 times slower than the other two.

0.6.3 Accuracy

```
In [15]: plt.bar([i for i, _ in enumerate(model_labels)],
                 [log_met[0], svm_met[0], for_met[0]])
         plt.title('Model Accuracy by Model')
         plt.ylabel('Accuracy (% 0.0 - 1.0)')
         plt.xlabel('Model')
         plt.xticks([i for i, _ in enumerate(model_labels)],
                    model_labels)

         plt.rcParams['figure.figsize'] = [12, 6]
         plt.show()
```



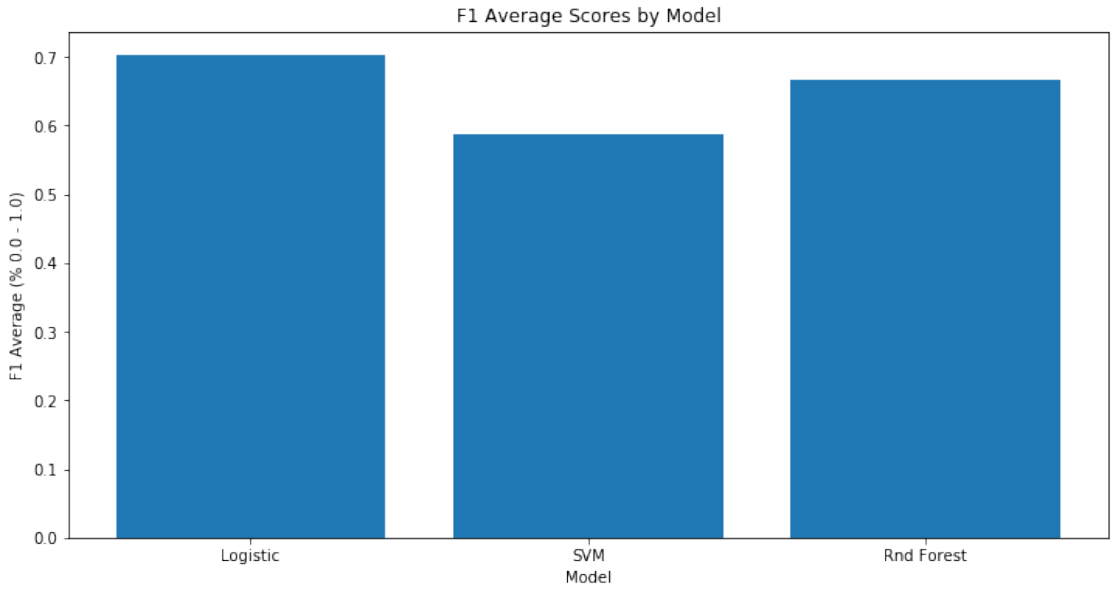
The accuracy of the three seems close, but the logistic regression seems to be the best.

0.6.4 F1 Scores

```
In [22]: plt.bar([i for i, _ in enumerate(model_labels)],
                 [log_met[2], svm_met[2], for_met[2]])
         plt.title('F1 Average Scores by Model')
         plt.ylabel('F1 Average (% 0.0 - 1.0)')
         plt.xlabel('Model')
         plt.xticks([i for i, _ in enumerate(model_labels)],
                    model_labels)

         plt.rcParams['figure.figsize'] = [12, 6]

         plt.savefig('../reports/figures/f1-score.png') # save figure
         plt.show()
```



The average F1 scores are comparable between the Random Forest and the Logistic model, but the SVM model seems to perform noticeably worse than the other two.

0.6.5 Summary of Model Assessment

When it comes to classification performance (f1 and accuracy), the logistic regression seems to best the two other models, with the the SVM model particularly performing worse. However, these scores could have been effected greatly by the tuning, as the SVM model was untuned and the Random Forest had a narrow set of tuning parameters due to runtimes. Moreover, the skin lesion conditions frequencies are very unequally divided between classes making f1 scores difficult to interpret.

For run times, it seems that the Random Forest seems to take the overall highest run time due to its long fitting - probably caused by the tuning. The SVM model seems to be very slow when it comes to predictions - which is problematic considering it was not even tuned. The SVM performance is no surprise however, considering its worse than quadratic complexity ("sklearn.svm.SVC — scikit-learn 0.20.2 documentation," n.d.). The logistic model seems to be significantly faster than the two overall according to the combined fit and prediction times.

Nonetheless, these three models were quite difficult to use with this dataset. As the Logistic Regression does not seem to perform well when it comes to its confusion matrix, and the other two models come with significant run time issues and do not improve on its performance - because said run time issues make it difficult to tune them!

Therefore, while out of the three models the logistic regression is best, for problems with high dimensionality it might be wise to use a method that is more paralyzable (e.g. Neural Networks) to make tuning easier with Parallel processing based Graphics Cards.

0.7 Evaluation of Data Mining Process

0.7.1 Business Success Criteria and Future Work

- Develop a neural, logistic, SVM and/or random forest models
  - While a neural network was not developed due to time constraints the other models were fitted and predicted with.
- Train the models using a diverse and large sample sized dataset
  - While HMINST dataset seems to have a large selection of conditions, the frequencies of some conditions are minimal.
- Classify the diagnosis based on a selection of skin conditions (stated above)
  - While the overall accuracy of the models was good, the false positive and negative performances were poor.
- Find and analyse model confusion matrices and measure models accuracy (discriminatory)
  - Confusion matrices were printed and accuracies shown alongside f1 scores.
- Find and analyse model fit training speed and test/usage speed (non-discriminatory)
  - Training and test(predictions) times were recorded and analysed.
- Select a favourable model based on the evaluation measures, but prioritise good false negative performance
  - Logistic Regression was selected as the better model due to its good run time and it had the best false negative performance according to its f1 scores. However, its false negative performance is still poor with some classes.

Overall, as stated in the model assessment summary, the models used proved to be problematic, especially the SVM and the Random Forest models, which proved to be slow and difficult to use and tune. Moreover, the dataset did not cover all the skin lesion conditions with sufficient frequencies. Hence, any future work should strive to try a more paralyzable model that is easier to tune and attempt to make use of a dataset with better frequency balance if possible.

0.7.2 Techniques and Tools

While this project carried over a lot of familiar tools and processes used in previous projects (e.g. CRISP-DM, Git), which proved to proved the base process and the version control backup/archive for this project well, there was major change to new environment.

This project was undertaken over a Cookie Cutter Data Science template over a python based environment, an environment which while successfully provided new more controllable tools (e.g. virtual environments for isolation), it also proved to be challenging. As, unlike R based solutions used before like Project Template, some solutions in this environment like the Cookie Cutter templates require more hacking (e.g. no automatic dataset loading from .csv files). The extra hacking proved to be a hindrance in the project and might have lead to some delays, however it provided useful tools that will be used in future projects.

Lastly, an issue was faced in this project regrading the CRISP-DM model followed. In previous projects, the Data Preparation stage was done before the Data Understanding stage, but when used

in this project the lack of clarity on the dataset lead to some parts of the project taking too long (e.g. merges due to key issues) due to a poor understanding of the data. Hence in future projects a slimmed down Data Understanding stage might be added before the data preparation.

0.8 References

1. James, G., Witten, D., Hastie, T., Tibshirani, R., 2013. Classification, in: James, G., Witten, D., Hastie, T., Tibshirani, R. (Eds.), An Introduction to Statistical Learning: With Applications in R, Springer Texts in Statistics. Springer New York, New York, NY, pp. 127–173. https://doi.org/10.1007/978-1-4614-7138-7_4

2. Rieck, K., Sonnenburg, S., Mika, S., Schäfer, C., Laskov, P., Tax, D., Müller, K.-R., 2012. Support Vector Machines, in: Gentle, J.E., Härdle, W.K., Mori, Y. (Eds.), Handbook of Computational Statistics: Concepts and Methods, Springer Handbooks of Computational Statistics. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 883–926. https://doi.org/10.1007/978-3-642-21551-3_30

3. sklearn.multiclass.OneVsRestClassifier — scikit-learn 0.20.2 documentation [WWW Document], n.d. URL https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html (accessed 1.24.19).

4. sklearn.ensemble.BaggingClassifier — scikit-learn 0.20.2 documentation [WWW Document], n.d. URL https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html (accessed 1.24.19).

5. 3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.2 documentation [WWW Document], n.d. URL https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (accessed 1.24.19).

6. 1.10. Decision Trees — scikit-learn 0.20.2 documentation [WWW Document], n.d. URL https://scikit-learn.org/stable/modules/tree.html#tree-classification (accessed 1.24.19).

7. pjh2011, 2018. Contribute to pjh2011/rf_perm_feat_import development by creating an account on GitHub.

8. sklearn.svm.SVC — scikit-learn 0.20.2 documentation [WWW Document], n.d. URL https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (accessed 1.24.19).