Predicting Faults in Steel Plates

GURU MANIKATNA INNAMURI
DEPT. OF CECS (DATA SCIENCE)
UNIVERSITY OF MICHIGAN, DEARBORN
DEARBORN, UNITED STATES OF AMERICA

***Abstract*— The following is a paper on methodology to accurately predict the kind of fault(s) in steel plates using Machine Learning techniques. The data for this problem has been collected from images of various steel plates with certain properties. These attributes are then used to determine the kind or fault(s) observed in a given plate. Broadly, there are 7 fault types relevant to the problem at hand. More than one faults can occur in any given plate and the faults can also be correlated to each other. This kind of problem can be categorized a multi-label problem. The idea of this paper is to determine correlations, see if all or only a set of attributes are enough to achieve good accuracy and a fair balance between accuracy and f1 score. To achieve the aforementioned goal, a series of problem formulation, data preprocessing, modeling steps are followed. This paper is structured as follows: problem formulation, data understanding, preprocessing and model building however, the flow could be recursive. For the purpose of this problem, methods available on Python (pertaining to the above steps) are utilized wherever possible**

Keywords*— *multi-label modeling, steel plates, prediction, Python, preprocessing

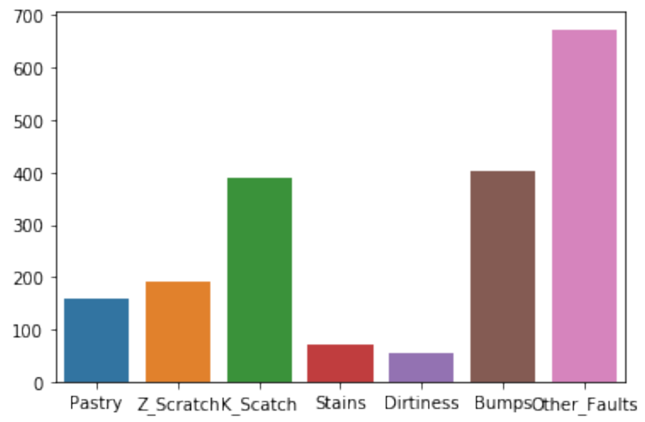
CONTENTS

I. INTRODUCTION

A multi-label classification problem is one in which target variable can take on multiple values. Each data point from X, the input variables can get 1 or more values as output. The problem at hand is to predict the type of faults each given steel plate could have depending on the given inputs including values like luminosity, pixels area, type of steel among others. The design matrix for this problem would be all the given attributes as X and the kinds of faults as Y(target) which in this case consists of 7 types. The target is the difference between a simple classification problem and multi-label problem. There are multiple methods to solve this problem that consider the correlations between targets, considers each target variable as an independent entity or even build a new feature out of the existing target that would represent the combinations of targets. In the dataset provided by [1] there are a total of 34 features out of which 27 are the attributes that can help determine the kind(s) of faults and the rest are faults themselves. To solve this problem, I have used the approaches BinaryRelevance, Algorithm Adaption, and Classifier Chain. At a very high level, these methods try to extract the patterns from all the possible target values using the algorithm that is selected. Throughout the course of this experiment, various models have been built, combinations of scaling, feature selection, extraction were used to improve model performances. The metrics used for this purpose were Accuracy and F1 score. These metrics can't be directly used as is since the number of labels are more than 2, we need to specify the usage of micro or macro averages. For the sake of this problem I have considered micro average since I am interested in the performance of models on individual faults as well. From here on the metrics accuracy and F1_score refer to the versions that use micro averaging

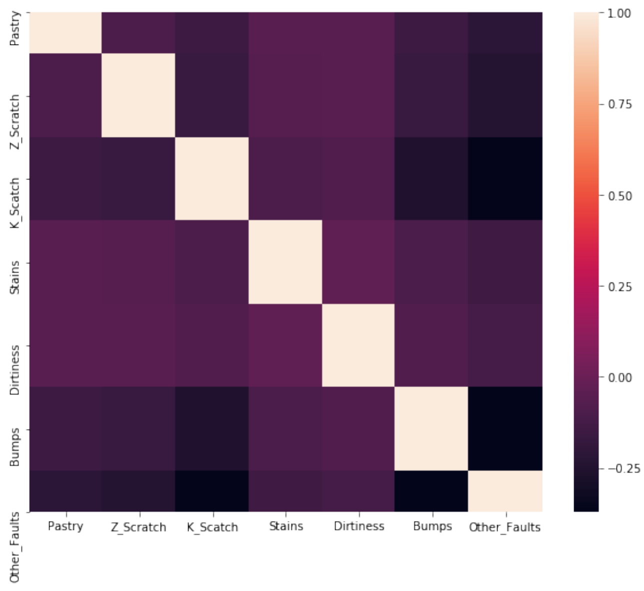
Models with a balance between the 2 metrics are clear winners. In my case, the Support Vector Classifier and Logistic Regression have turned out to be the winners. Random Forest Classifier also had a good accuracy and F1 but there seems to be overfit.

II. VISUALIZATION



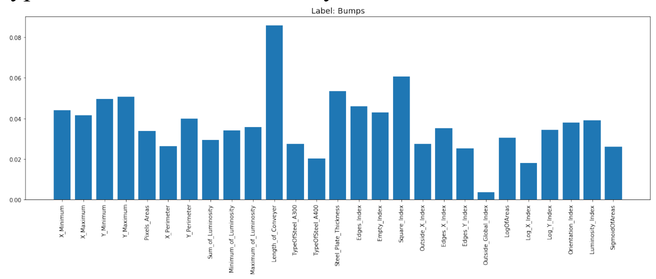
Above is the frequency graph for the types of faults. It is very evident that the faults Stains and Dirtiness have lot less entries in comparison to the others. This fact was noted and indeed turned out to be useful while identifying the most important features.

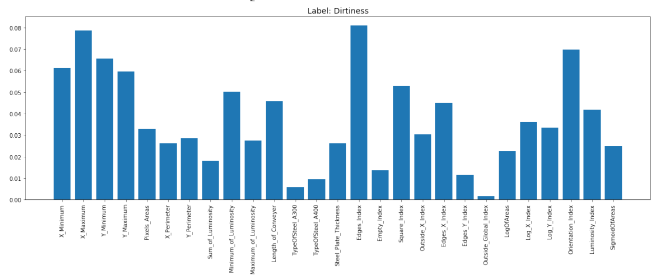
It is also worthy to take note that the target feaures themselves might have correlation amongst themselves. A plot had been made to identify how the targets correlate to each other. It was found that there are some that show considerable correlation. This fact had been important to note since it led to realizing the necessity of applying chaining while building models. Below is the correlation plot for the targets.

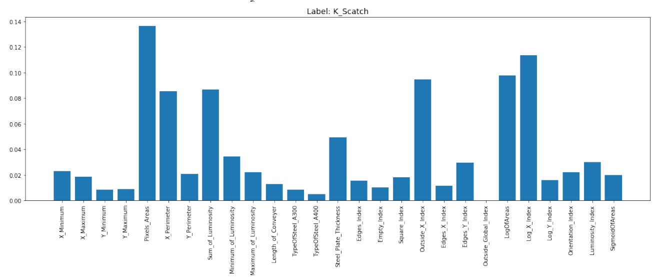


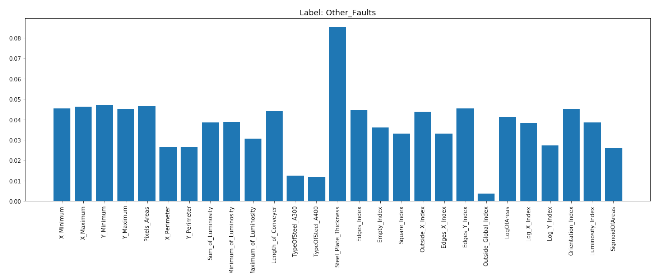
The correlations were not very high to be considered strong but taking into account these correlations had improved the performance a little bit as will be seen later.

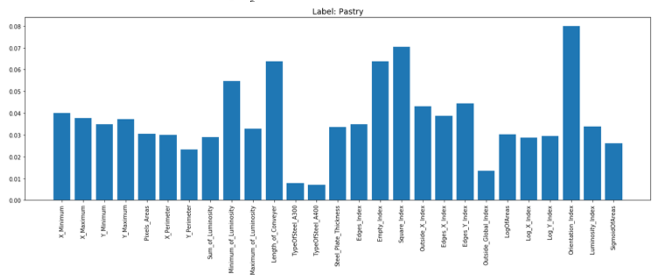
To identify the relevant features, a RandomForestClassifier had been built on each of the fault types seperately and the respective feature importances have been ploted with an idea of observing patterns in the importances and the variables that had significant importance in predicting each of the fault types could be considered truly useful.

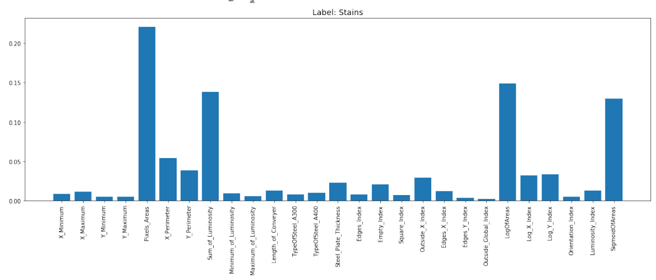


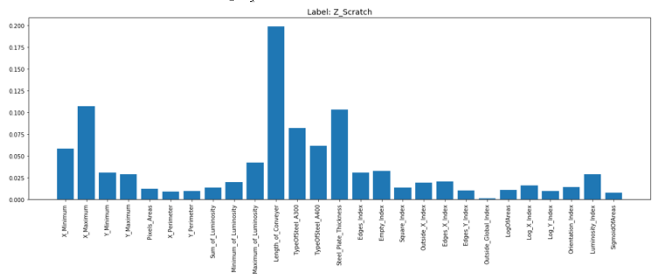








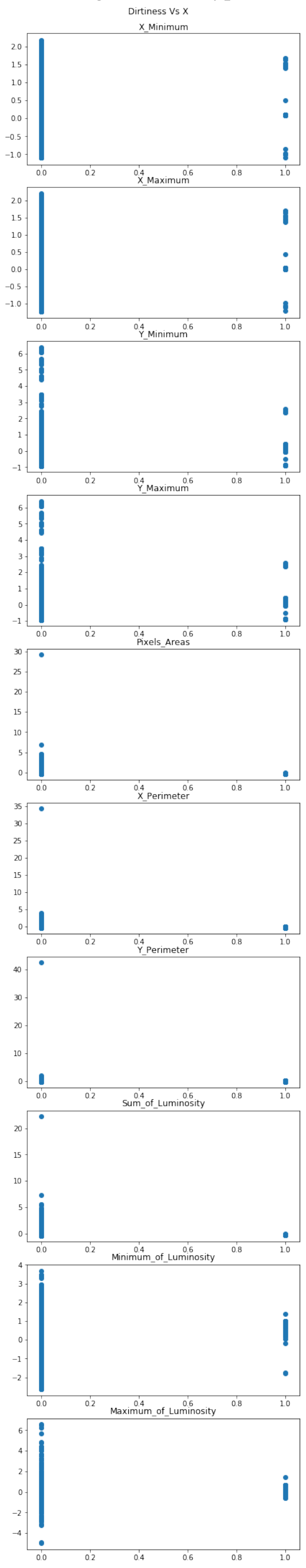


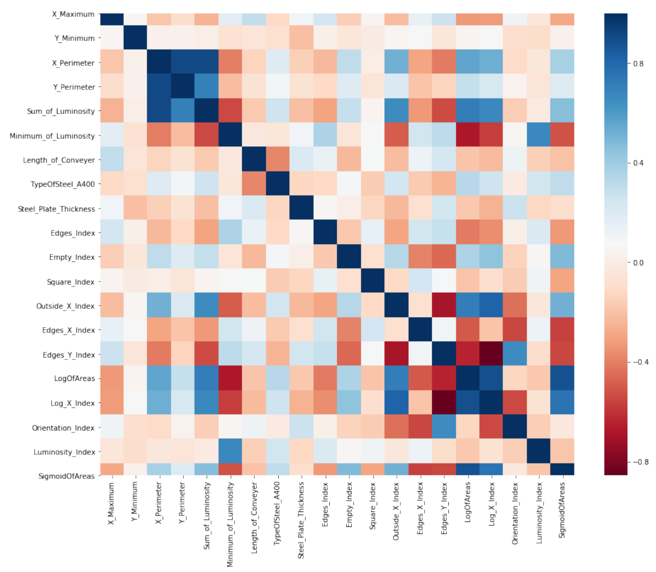


There wasn’t enough proof to say that one feature is more important than another from the above plots. So, using the feature correlations and the relation with each feature to each of the fault types, I was able to remove 7 features: Log_Y_Index, Outside_Global_Index, X_Minimum, Y_Maximum, TypeOfSteel_A300, Maximum_of_Luminosity, and Pixels_Areas. To have a standard for comparison, I have run the model on all 27 columns first. The logistic model without any preprocessing, tuning and all 27 features intact had accuracies 19% and 21% on train and test sets respectively. Simply removing the features had only helped so much so to increase the accuracy to around 50% on train data and 40% on test data. However, after some more preprocessing and tuning, this model with reduced features turned out to be one of the best models.

As shown below, each feature had been plotted against each fault type to see if there is a difference in variance that could be captured. Since there were too many plots to it, only a snippet is being shown. For a lot of features, there were fault types that didn’t have any variation in positive and negative groups. Taking in to account those and the correlations of the features themselves 7 features were removed as aforementioned.

Below are the plots for X correlation after feature removal and snippet of features against fault types.





After removing the features as said earlier, the multi-correlation had reduced.

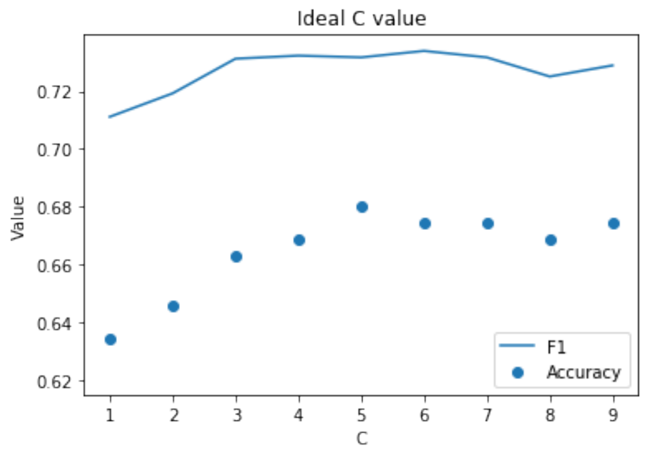
III. Methodology Development

The given problem can be solved in various ways out of which I have had the opportunity to try out 3.

a) Binary Relevance:
   a. It is a method very similar to one vs rest, but without consideration to correlation among labels
   b. Training set is fit on each label independently
b) ClassifierChain:
   a. Unlike Binary Relevance, this method also takes into account the effect of correlation among features
   b. Train set is first trained on one label and then the label is included in the X to train next label hence accounting for relations between labels
c) Algorithm Adaption:
   a. These are the algorithms that are fabricated to work with multi-label data without any modifications to the dataset itself

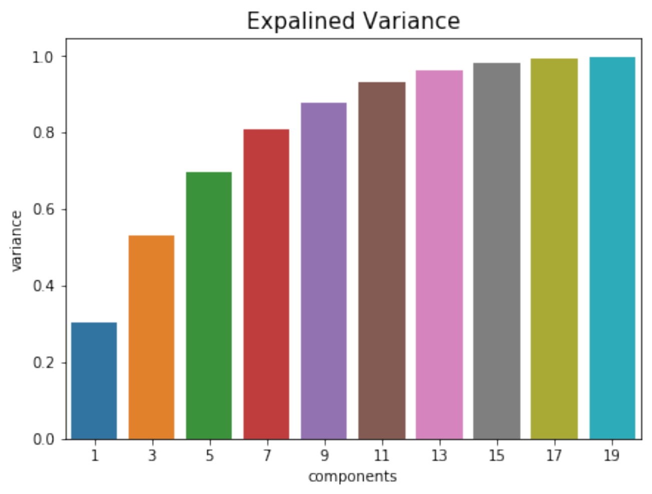
Binary Relevance and Classifier Chain both belong to a family of multi-label classification called Problem Formulation. In this family, the algorithms are directed toward modifying the problem statement to fit data.

The process started out with training models on the raw data without any preprocessing. First to identify features of importance using Random Forest. During this stage itself, the RandomForest classifier had overfit the data with train accuracy and F1 scores of 100% and when tested on the test data it got down to 77% and 77% respectively. This clear overfit encouraged me not to go ahead with a complex model right from the beginning. So, I worked back various different models, tweaked some parameters but couldn't achieve scores any better than 60% without overfit. It is note-worthy that the initial SVC gave a test performance of 20% accuracy. From there, I scaled the data since SVC works better with scaled data and achieved a performance of around 64% with test data. I then went on to tune the SVC model for an increased performance of 67% accuracy on the test set. Below are the tuning particulars of the SVC.



Above is the validation accuracy and F1 for the SVC trained on scaled data. This is by far the first model to reach a test score above 65%. However, this improvement in performance isn't just because of tuning, before getting to tuning, I had also applied PCA on the data to reduce the number of features to 15

To this point all the work done on SVC had only been using Binary Relevance.



Validation models have been built with number of components 15,17 and 19 and there isn't enough increase in performance so as to retain 19 features. Hence going forward any discussion regarding PCA would mean the above 15 principle components.

Then a Random Forest had been trained on the above 15 principle components using a Classifier Chain. This had reduced the overfit encountered by RandomForest initially. The gap between train and test accuracies have now reduced to almost half of what was seen in a model without any preprocessing. Overall, tuning the RandomForest model(on validation) had resulted in an increase of 2% in test accuracy.

	Model	Method	Processing	Data	Accuracy	F1_score
6	RandomForest	ClassifierChain	None	Train	1.000000	1.000000
7	RandomForest	ClassifierChain	None	Test	0.774359	0.774359
8	RandomForest	ClassifierChain	PCA	Train	0.825589	0.825852
9	RandomForest	ClassifierChain	PCA	Test	0.702564	0.704370
10	RandomForest	ClassifierChain	PCA and tuned	Train	0.860598	0.862058
11	RandomForest	ClassifierChain	PCA and tuned	Test	0.723077	0.728205

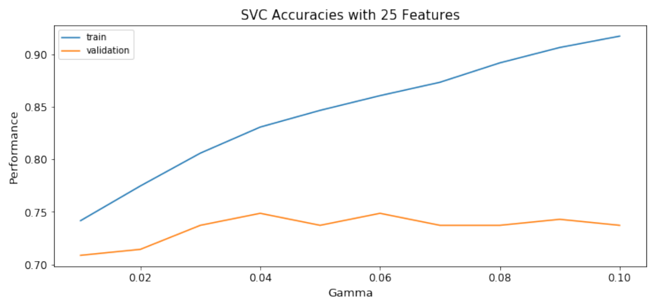
Since SVC had been the best performer amongst all the Binary Relevance formulated models, I have then tried Classifier Chain on SVC as well since I wanted to check how the correlation between labels is paying towards the model performance. So, I trained a ClassifierChain SVC on the scaled and PCA applied set with some tuning to end up with a great performance of above 70% test accuracy for the first time after starting this project.

I then tried out the multi-label supporting algorithm MLKNN which doesn't require the data to be transformed in any way. We simply have to pass the X and Y(with multiple columns) as is. However, some preprocessing helped it get to speed but in the end it couldn't compete with the likes of SVC and the maximum performance it could attain was 69% test accuracy.

After this point, the best thing to do was to try and improve the performances of the existing best models and not try new ones. In that spirit, I fell back on feature selection once again and used SlectKBest technique from sklearn to select important features from the given set. The results of that task are shown below:

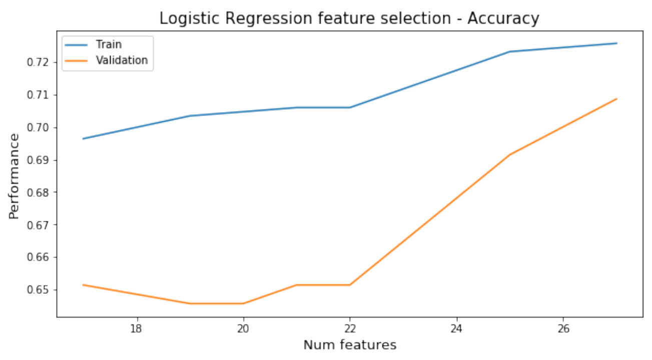
	num_cols	train_accuracies	train_f1	validation_accuracies	validation_f1
0	17	0.723106	0.723106	0.640000	0.640000
1	19	0.737747	0.737747	0.668571	0.668571
2	19	0.737747	0.737747	0.668571	0.668571
3	20	0.739656	0.739656	0.680000	0.681948
4	21	0.737110	0.737110	0.685714	0.687679
5	21	0.737110	0.737110	0.685714	0.687679
6	21	0.737110	0.737110	0.685714	0.687679
7	22	0.737110	0.737110	0.685714	0.687679
8	25	0.763208	0.765156	0.714286	0.716332
9	25	0.763208	0.765156	0.714286	0.716332
10	27	0.772120	0.774091	0.725714	0.729885

In the SelectKBest method, the feature importances of each feature with respect to each of the labels are returned. I have averaged the importance of each feature and selected the columns that have an avg. importance greater than a given threshold which started at mean average importance and reduced with each iteration. As seen above, the best score was for 27 features but since the difference isn't much I have selected 25 features and tried to tune the parameters.



The validation accuracy was highest at gamma 0.04 but still couldn't beat the test accuracy of previous best SVC which stood at 75.3%

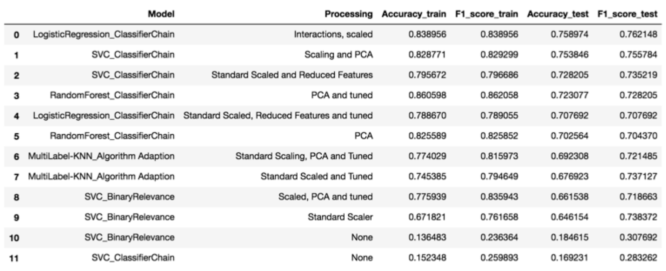
I have applied the same SelectKBest on LogisticRegression as well and the results are as follows on validation set



The test performance was also in the similar range of the validation and hence I dropped this idea.

Then I have removed the 7 features that I discussed in the beginning and tried it on SVC, Random Forest and Logistic Regression. Out of these Logistic had the best perforamance. This best performance didn't come entirely from the feature removal. It went step by step. Using KBest gave a performance of around 66% test accuracy, using Scaling and 7 features removed increased the test score to 69%. Over this, I have included polynomial interactions between the X variables which resulted in the best logistic model in my project at 75% accuracy.

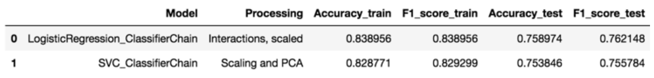
IV. Results



	Model	Processing	Accuracy_train	F1_score_train	Accuracy_test	F1_score_test
0	LogisticRegression_ClassifierChain	Interactions, scaled	0.838956	0.838956	0.758974	0.762148
1	SVC_ClassifierChain	Scaling and PCA	0.828771	0.829299	0.753846	0.755784
2	SVC_ClassifierChain	Standard Scaled and Reduced Features	0.795672	0.796686	0.728205	0.735219
3	RandomForest_ClassifierChain	PCA and tuned	0.860598	0.862058	0.723077	0.728205
4	LogisticRegression_ClassifierChain	Standard Scaled, Reduced Features and tuned	0.788670	0.789055	0.707692	0.707692
5	RandomForest_ClassifierChain	PCA	0.825589	0.825852	0.702564	0.704370
6	MultiLabel-KNN_Algorithm Adaption	Standard Scaling, PCA and Tuned	0.774029	0.815973	0.692308	0.721485
7	MultiLabel-KNN_Algorithm Adaption	Standard Scaled and Tuned	0.745385	0.794649	0.676923	0.737127
8	SVC_BinaryRelevance	Scaled, PCA and tuned	0.775939	0.835943	0.661538	0.718663
9	SVC_BinaryRelevance	Standard Scaler	0.671821	0.761658	0.646154	0.738372
10	SVC_BinaryRelevance	None	0.136483	0.236364	0.184615	0.307692
11	SVC_ClassifierChain	None	0.152348	0.259893	0.169231	0.283262

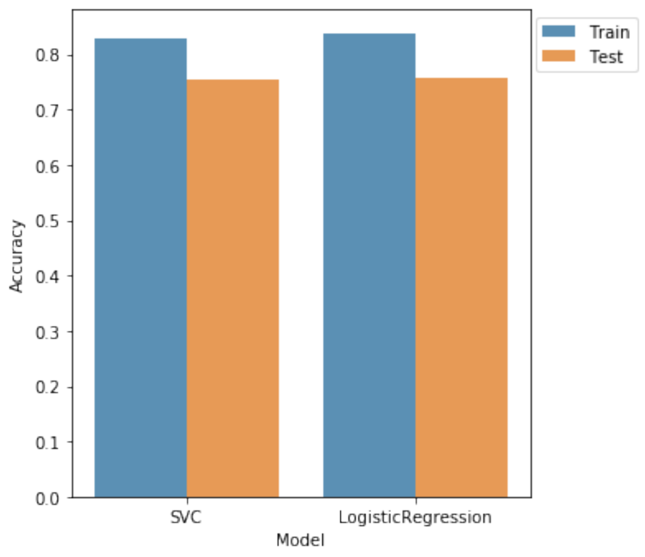
The best models after considering the overfit, balance between accuracy and F1 scores in this whole exercise are SVC – Binary Relevance and Logistic Regression – Classifier Chain

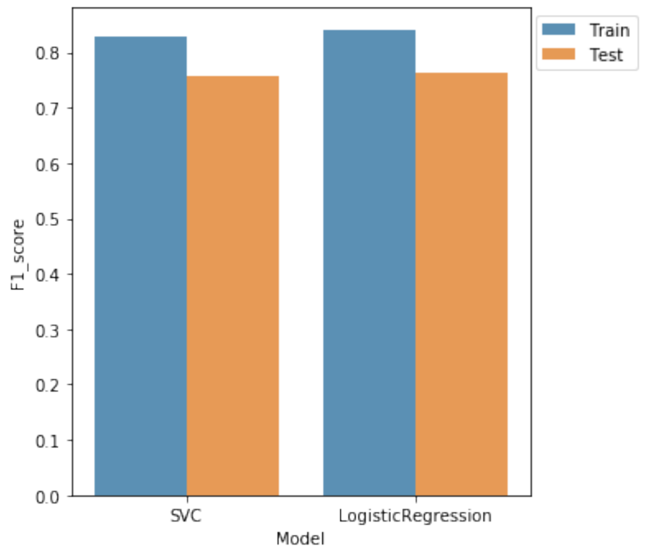
Final Scores:



	Model	Processing	Accuracy_train	F1_score_train	Accuracy_test	F1_score_test
0	LogisticRegression_ClassifierChain	Interactions, scaled	0.838956	0.838956	0.758974	0.762148
1	SVC_ClassifierChain	Scaling and PCA	0.828771	0.829299	0.753846	0.755784

Below are the results for the best models over all. SVC with binaray relevance and Logistic with Classifier chain along with polynomial feature interactions.





V. Conclusion

Interactions played a major role in predicting the faults in given steel plates. Though various methods were tried for feature selection, the best one turned out to be hand selection. Recursive feature elimination, SelectKBest, RandomForest were othere methods tried for feature selection but with no luck. There still seems to be scope for improving the performance of the above models. The future work can be focused on feature engineering, selection and preprocessing rather than more techniques to work on predictions themselves.

Random Forest model always had a competitive performance in comparision to the SVC and Logistic models. But the problem was overfitting. Even after removing features by hand the over fit still existed. Train performance on the dataset without hand removed features was 98% while the same for test reduced to about 72%. Over fit had been clearly visible since the start with Random Forests.

I hope to try advanced methods such as GradientBoosting trees, Neural networks among others in the future work to verify how they fair in this comparison for prediction performances.

VI. References & Appendix

[1] Dataset provided by Semeion, Research Center of Sciences of Communication, Via Sersale 117, 00128, Rome, Italy.

- Multi-Label Collective Classification Xiangnan Kong, Xiaoxiao Shi, Philip S. Yu, here

- Classifier Chains: A Review and Perspectives Jesse Read , Bernhard Pfahringer , Geoff Holmes , and Eibe Frank, here

- https://machinelearningmastery.com/multi-label-classification-with-deep-learning/

- https://en.wikipedia.org/wiki/Multi-label_classification

Appendix:

Code to train the models

```
def
train_classifiers(clf,X_train,X_test,y_train,y_test,label='test'
):
    clf.fit(X_train,y_train)
    train_pred = clf.predict(X_train)
    test_pred = clf.predict(X_test)

    print('train accuracy: {}, f1_score:
{}'.format(accuracy_score(y_train,train_pred),

f1_score(y_train,train_pred,average='micro')))
    print('{} accuracy: {}, f1_score:
{}'.format(label,accuracy_score(y_test,test_pred),

f1_score(y_test,test_pred,average='micro')))
```

The X_train, X_test could be any of the scaled X, validation X, reduced features X and the same goes for y_train and y_test as well. The differentiating factor between test and validation is the label parameter. When I passed validation data, I have changed the value of label to 'validation' and hence the results would be formatted without confusion.

Code to use SelectKBest:

```
for best in range(9,28,3):
    y_pred_new_train = pd.DataFrame(columns = y.columns)
    y_pred_new_val = pd.DataFrame(columns = y.columns)
    for fault in y_train_scaled.columns:
        svc_cc = SVC(C=3)
        kbest = SelectKBest(k=best)
        X_train_new_scaled =
kbest.fit_transform(X_train_scaled,y_train_scaled[fault])
        X_val_new_scaled = kbest.transform(X_val_scaled)
        svc_cc.fit(X_train_new_scaled,y_train_scaled[fault])
        y_pred_new_train[fault] =
svc_cc.predict(X_train_new_scaled)
        y_pred_new_val[fault] =
svc_cc.predict(X_val_new_scaled)

    print('Best {} features'.format(best))
    print('Train accuracy: {}, F1:
{}'.format(accuracy_score(y_train_scaled,y_pred_new_train
),

f1_score(y_train_scaled,y_pred_new_train,average='micro')
))
    print('Validation accuracy: {}, F1:
{}'.format(accuracy_score(y_val_scaled,y_pred_new_val),

f1_score(y_val_scaled,y_pred_new_val,average='micro')))
    print('\n')
```

The above function coupled with the following one would give the performances on validation set for various number of features selected

```
def identify_imp_features(clf):
    temp = {'thresholds': np.arange(5,-0.5,-0.5),
            'num_cols': [],
    'train_accuracies': [],
    'train_f1': [],
    'validation_accuracies': [],
    'validation_f1': []
    }
    for th in temp['thresholds']:
        selected_cols =
X_train.columns[(np.where(np.mean(selected_features,axis
=0) >= th)[0])]
        num_cols = len(selected_cols)
        temp['num_cols'].append(num_cols)

        lgreg =
ClassifierChain(clf)#LogisticRegression(solver='liblinear',p
enalty='l1'))
        lgreg.fit(X_train_scaled[selected_cols],y_train_scaled)
        pred = lgreg.predict(X_train_scaled[selected_cols])
        pred_test = lgreg.predict(X_val_scaled[selected_cols])


temp['train_accuracies'].append(accuracy_score(y_train_sca
led,pred))

temp['train_f1'].append(f1_score(y_train_scaled,pred,averag
e='micro'))

temp['validation_accuracies'].append(accuracy_score(y_val
_scaled,pred_test))

temp['validation_f1'].append(f1_score(y_val_scaled,pred_te
st,average='micro'))

    perf = pd.DataFrame(data = temp)

    return perf
```

The above codes from the appendix were the most important pieces of code in the project. However, they aren't the only important ones. Polynomial features, tuning for thresholds, tuning model parameters and visualizations have also played a key role, but the code hasn't been included since it's too exhaustive.
Follow link for full code: https://tinyurl.com/faultPrediction