Prediction of Insurance Costs

1st Murat Biberoğlu
Computer Engineering
Istanbul Technical University
Istanbul, Turkiye
biberoglu20@itu.edu.tr

2nd Ömer Yıldız
Computer Engineering
Istanbul Technical University
Istanbul, Turkiye
yildizom20@itu.edu.tr

3rd Süleyman Ceyhun Demir
Computer Engineering
Istanbul Technical University
Istanbul, Turkiye
demirsu21@itu.edu.tr

I. Introduction

In this project, we were given a data set consisting of images that belong to 10 different classes. This images feature data were provided with 4 feature sets that were extracted via CLIP, DINO, ResNet and ViT. The train and test values were already separated by `train_feats.npy` for training and `valtest_feats.npy` to test the developed model on and submit to kaggle. Firstly, the train features were separated by their extraction method since each of the extraction methods has its strengths and weaknesses, they are all unique thus if we were to concatenate them before preprocessing we would risk losing these unique qualities. After the separation normalization and dimensionality reduction methods were applied (LDA and Nystroem). Then a Custom Stacking Classifier was applied for our model which uses predictions of multiple models to combine them into one prediction. Lastly 5-fold cross validation was applied with this Custom Stacking Classifier to get the definitive result. Our group BYD was placed 14th with the F1 score of 0.9899014 on the leaderboard at the end.

II. Datasets

Before sending the data input to the model, preprocessing was completed in a few stages.

A. Feature Set Grouping

In the first stage separately extracted features are grouped within themselves to prevent losing the strengths of the separate extraction models during further stages. When the feature selection was applied with all the feature sets in one place rather than within themselves we had significantly worse results, proving its important not to let different methods features overwhelm others. This is achieved by (Avoided including the same code for the validation set):

```
train_data_dict=train_features.item()
resnet_features=train_data_dict["resnet_feat
clip_features=train_data_dict["clip_feature
dino_features=train_data_dict["dino_feature
vit_features=train_data_dict["vit_feature"]
...
```

The code above for both train and validation features.
Then the labels are added to the data from `train_labels.csv` by merging on the ”ID” column of the csv file and ”idx” column of `train_data_dict`. All separate features are normalized within themselves.

```
clip_features=scaler.fit_transform(clip_features)
dino_features=scaler.fit_transform(dino_features)
vit_features=scaler.fit_transform(vit_features)
...
```

B. Feature Selection

And then LDA transformation is applied on to the sets.

```
def perform_lda(features,
    labels, n_components=None):
    lda = LinearDiscriminantAnalysis(
            n_components=n_components)
    reduced_features = lda.fit_transform(features
                , labels)
    return reduced_features, lda
n_components = 9
resnet_lda_features, lda_resnet = perform_lda(
            resnet_features, labels, n_components)
...
```

Nystroem transformation is also applied on to the sets separately.

```
def perform_nystroem(features,
            n_components=250):
    nystroem = Nystroem(kernel='rbf',
            n_components=n_components
                , random_state=42)
    reduced_features =
            nystroem.fit_transform(features)
    return reduced_features, nystroem
n_components = 27
resnet_nystroem_features,nystroem_resnet
    =perform_nystroem(resnet_features, n_component
...
```

Nystroem was applied due to its compatibility with kernel-based learning models along with Nystroem LDA is also applied to ensure if the data is separated more linearly those features will be also kept inside our dataset. Experiments made showed these two cultivated the best results among the different methods that were utilized.

C. Merging

Lastly all different sets in hand are combined into one dataset to use in the model.

```
lda_features_combined =
    np.hstack([resnet_lda_features
    ,clip_lda_features, dino_lda_features
    , vit_lda_features])
lda_features_combined.shape

nystroem_features_combined =
    np.hstack([resnet_nystroem_features
    ,clip_nystroem_features
    ,dino_nystroem_features
    , vit_nystroem_features])
nystroem_features_combined.shape

combined_features =
        np.hstack([lda_features_combined
        , nystroem_features_combined])
combined_features.shape
```

III. Methods

A. Figure Illustrating the Pipeline

The image is at the last page to not disrupt the page flow.

B. Random Forest Classifier

Random forest is a model that builds multiple decision trees, trees are built from a random subset of data and at nodes for these trees only a random few features are examined for splitting. After building with a set number of trees with set maximum depths each row generates a prediction from each tree and majority vote of the trees gives us the label prediction of the model. It is resistant to overfitting and very versatile although for large datasets its expensive to calculate. We utilized a Random Forest classifier with 300 decision trees and with maximum depth of 30.

C. K- Nearest Neighbors Classifier

It has all points in memory and for each new node it calculates the distance of them to the new node and selects the k nearest ones to decide which class this node belongs to by assigning the class of the most numbered neighbors. It has bad performance on high dimension data but is flexible and simple. The K- Nearest Neighbors classifier we used looks at 19 nearest neighbors to calculate the class of a certain data point.

D. Linear Support Vector Classifier

This model assumes that the data we have is linearly separable. Then finds the hyperplane that has the distance to the nearest points to of the separated classes. It decides the class of the given point according to which side of the hyperplane the given point lands on. It works pretty well with data that has higher dimensions although it works bad if the data is not linearly separable. It does not have any metrics.

E. Stochastic Gradient Descent Classifier

Selects a few training samples to compute the gradient from. Then updates the model weights according to the direction of minimizing the loss function. Repeats this for the given iteration count until convergence. Our Stochastic Gradient Descent classifier is set up to run up to 5000 iterations and stop early if the loss doesn't improve beyond a tolerance of 10 to the power of -8.

F. Logistic Regression Classifier

It computes the weighted sum of the input, applies logistic function which maps output to 0 or 1 classifies it based on the threshold, by generating this for each classifier we can predict which class the given node belongs to. The logistic regression we implemented runs at max up to 25,000 iterations to ensure max convergence.

G. Custom Stacking Classifier

This custom function first gets the predictions for each entry with Random Forest Classifier, K- Nearest Neighbors Classifier, Linear Support Vector Classifier and Stochastic Gradient Descent Classifier then with the meta classifier which is a logistic regression function for each row it tries to predict out of each models prediction for that row a final prediction. And returns the labels. We selected this models since all the classifiers we used in this custom stacking classifier have distinct strengths and weaknesses by combining them and making another model learn from their predictions to predict a final value we reduce the bias of the model, decrease overfitting, handle more complex relationships better and its compatible when multiple extraction methods are used. We chose which models to use inside this stacking classifier based on our experiments with different models. Some of them were taking too long to compile and some of them resulted in worse results but this configuration resulted the most cost efficient solution.

Results and Conclusions

After applying 5-Fold Cross Validation to the Custom Stacking Classifier F1 Score achieved is: 0.9882257900304948. Our team BYD ranked as 14th and we had a score of 0.9899014.

References

[1] https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.StackingClassifier.html
[2] https://scikit-learn.org/1.5/modules/kernel_approximation.html

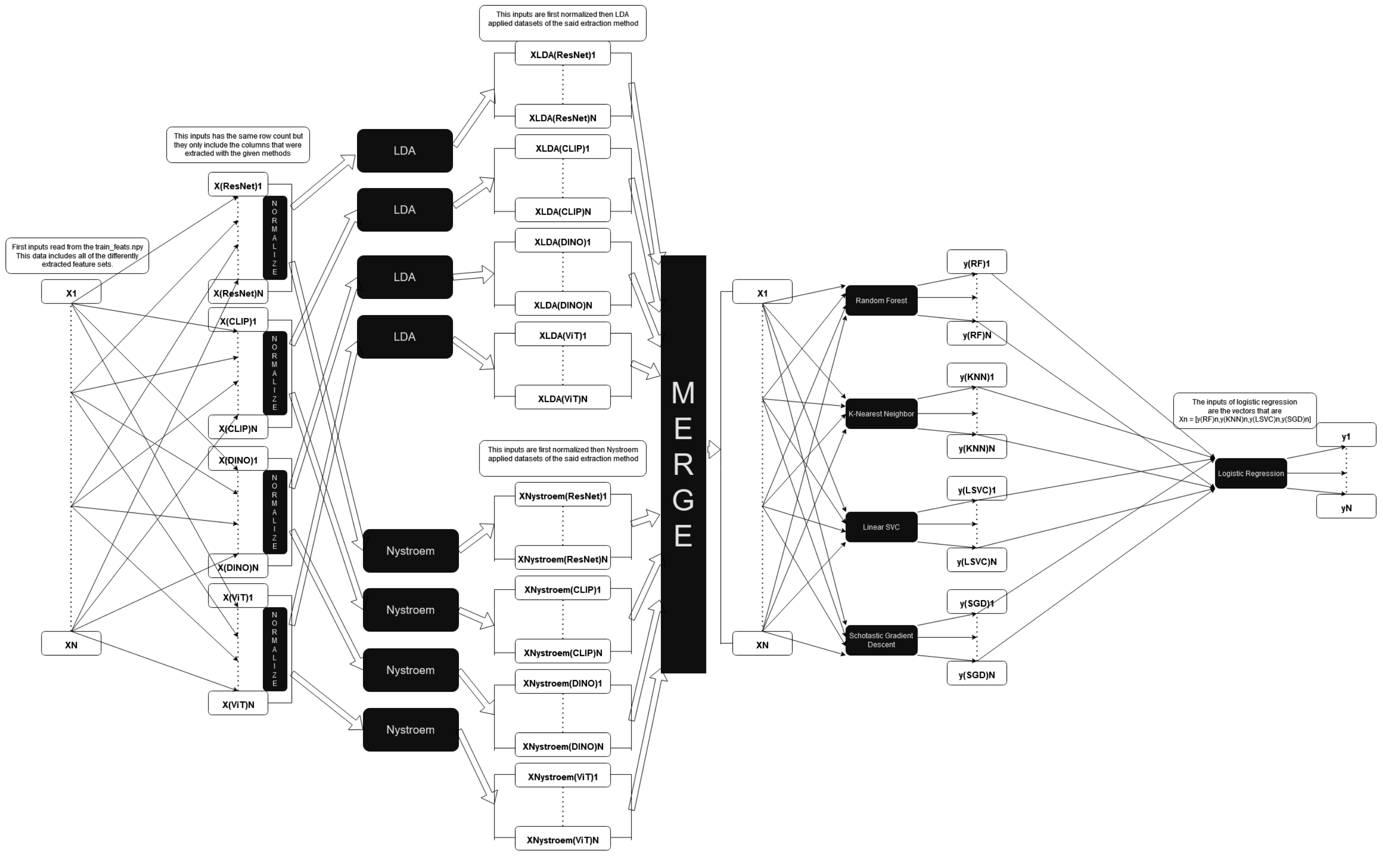
First inputs read from the train_feats.npy
This data includes all of the differently extracted feature sets.
X1
XN
This inputs has the same row count but they only include the columns that were extracted with the given methods
X(ResNet)1
X(ResNet)N
X(CLIP)1
X(CLIP)N
X(DINO)1
X(DINO)N
X(ViT)1
X(ViT)N
NORMALIZE
LDA
Nystroem
This inputs are first normalized then LDA applied datasets of the said extraction method
XLDA(ResNet)1
XLDA(ResNet)N
XLDA(CLIP)1
XLDA(CLIP)N
XLDA(DINO)1
XLDA(DINO)N
XLDA(ViT)1
XLDA(ViT)N
This inputs are first normalized then Nystroem applied datasets of the said extraction method
XNystroem(ResNet)1
XNystroem(ResNet)N
XNystroem(CLIP)1
XNystroem(CLIP)N
XNystroem(DINO)1
XNystroem(DINO)N
XNystroem(ViT)1
XNystroem(ViT)N
MERGE
X1
XN
Random Forest
K-Nearest Neighbor
Linear SVC
Schotastic Gradient Descent
y(RF)1
y(RF)N
y(KNN)1
y(KNN)N
y(LSVC)1
y(LSVC)N
y(SGD)1
y(SGD)N
The inputs of logistic regression are the vectors that are Xn = [y(RF)n,y(KNN)n,y(LSVC)n,y(SGD)n]
Logistic Regression
y1
yN

Fig. 1. Figure showing the pipeline.