CSCI 5922 Fall 2022–Lab 1

Abulitibu Tuguluke

September 11, 2022

The Data I used

Through the semester, I would probably used data generated from work, for this lab particularly, I used job posted from dice.com categorized tech job (e.g.: software engineer, data scientist) and non-tech job (e.g.: nurse, writer). It was carefully hand labeled by SMEs. I want to use Feed-forward network to predict whether a job post a tech job or not.

Table 1: Data Description

	text	accept	answer
count	968	968	968
unique	968	2	1
		Tech	accept
count	1	876	968
		Non-Tech	accept
count	1	92	968

Which stated there are 876 jobs labeled as **Tech**, and 92 as **NonTech**, which makes the total number of sample 968. Since I hand-picked the **accepted** answer, we only need two columns of information from this data-frame: **text** and **answer**. Here are the 2 examples:

- Non Tech (Job): Deliver with Uber Eats Deliver with Uber. Earn on your schedule. Work on your schedule. Deliver for a few hours in the mornings, every night, or just on weekends-it's up to you. Earn Good Money. You'll earn by bringing people the food they love from local restaurants. Choose your wheels. Use your car, scooter, or bike to make deliveries.* Delivery requirements Car delivery: Be at least 19 years old Have a 2-door or 4-door car made after 2000 Have a valid driver's license, vehicle registration and vehicle insurance Have at least one year of driving experience in the U.S. Bike delivery: Be at least 18 years old Have a state-issued ID or Driver's License When signing up be sure to choose 'Biking' under transportation method. Scooter delivery: Be at least 19 years old Have a valid driver's license and vehicle insurance Have a 2-wheel scooter made after 2000 Ready to get started? Sign up today and start earning. No experience necessary. But, if you have previous employment experience in delivery (such as a delivery driver, food service, food delivery, delivery runner, or courier) you may enjoy delivering with Uber Eats! *Vehicles allowed for delivery vary by city.

- Tech (Job): Hadoop Consultant In depth and strong knowledge and hands on experience Hadoop, Horton Works, Scoop, Oozie workflowsShould have done complex data ingestions in HadoopKnowledge in Data modeling and DWH naming patternData profiling knowledgeKnowledge in DWH and ETL concepts

Word embedding (Pre-processing)

Since the data is in text form, we need to convert it into numeric, **Tech** label is converted as 1, **Non-Tech** as 0. For description, we first used TF-IDF method, yet the reslut is not idea, as shown in Figure 1,

There are no major inprovement in the accuracy, my guess is that the TF-IDF vector is sparse, as we seen below:

```
[0.        , 0.07048313, 0.        , ..., 0.        , 0.        ,0.        ],
```

There is not much to learn, so we used another method similar to Word2Vec, called Doc2Vec, which embeds text into dense vector, e.g.:

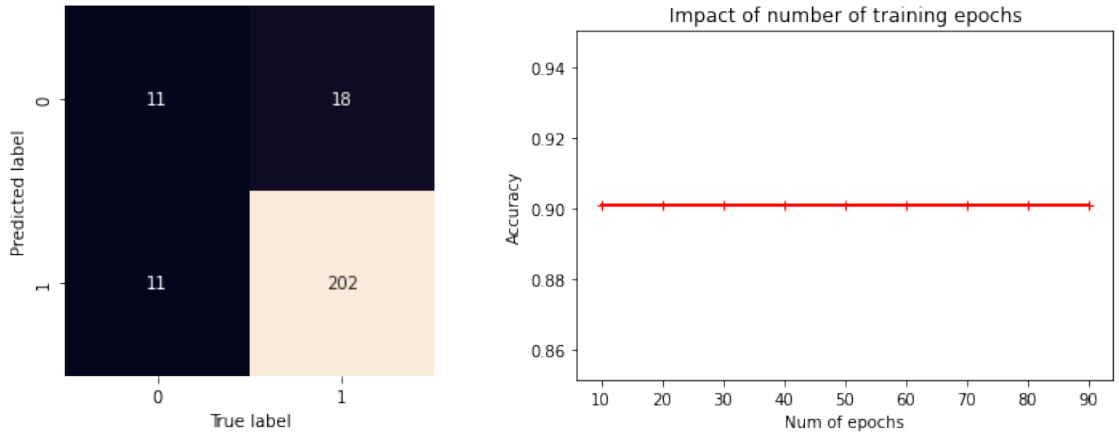

Figure 1: Tf-Idf emvedding for text.

```
array([ 0.69282454, -0.767151  ,  1.1698718 ,  0.0125812 ,  0.04911822,
-0.6022604 , -0.7765408 ,  0.14662024, -0.5277134 , -1.1415157 ,
0.9658156 ,  0.12996148, -0.7017618 ,  0.25222412,  0.2311553 ,
-0.4849749 ,  0.460774  , -0.6721754 ,  1.1252933 ,  0.5912803 ,
1.066204  ,  0.91162   , -0.13604069, -0.46164566, -0.22352642,
-0.02144009, -0.55692935,  0.12265404, -0.6255455 ,  0.02748431,
-0.05019109,  0.65613216,  1.3773457 ,  0.55956566,  0.5098652 ,
1.1547431 , -0.18142179,  0.1515532 , -0.4199732 ,  0.51574713,
0.3102894 ,  1.2909027 ,  0.5681546 ,  0.90260255, -0.82004976,
-0.76896405, -0.3467222 ,  0.12360989, -0.8401419 ,  0.09505583,
1.5170844 ,  0.887022  ,  0.88092905, -0.45412895, -0.07270601,
0.00651041,  0.0348754 ,  0.5411873 ,  0.07916429, -0.17342937,
0.39807764, -0.91081697, -0.55455333, -0.6905511 ], dtype=float32)
```

with which we will be using in this assignment.

1 Neural Network Hyper-parameters

We ran 9 experiments: 3 activation functions of **ReLU, Tanh, Logistic(Sigmoid)**, with 3 sets of hidden layers of $\mathbf{10 \times 10, 20 \times 20, 30 \times 30}$. We set rest of the Hyper-parameters as default in MLPClassifier, such as: batch size=200, learning rate=0.001, max iteration=200, optimization method=adam. The result is shown is Figure 2:
According to the accuracy data, all 3 layers with **ReLU** performs the best with 0.92 accuracy even with less neuron layers. **Tanh** is the least accurate with 0.90 after all three layers. But what I found more interesting is the True-Positive for **Non-Tech(0)** for **Sigmoid**, since the first 2 set of layers of $\mathbf{10 \times 10, 20 \times 20}$ did not able to spot a single **Non-Tech(0)** correctly until $\mathbf{30 \times 30}$, I suspect is it due to the high cost of non-linearity of the logistic functions which demands more computing to get to a better result. So my bets would be on **Tanh** performs better on this particular data set due to it's ability to label **Non-Tech(0)** correctly than **Sigmoid**, which confirms the textbook's claim that **ReLU** is the most popular activation function, for now.
Also, if we ignore the activation function, and solely focused on Layer architecture, then the bigger layer the better results, $\mathbf{30 \times 30}$ performs the best, which again confirms the book's claim that more computing yields better result. The Evaluation Metrics in Table 2 tells a similar story, but if we focus more on **Non-Tech(0)**'s precision, then clearly **ReLU** with 20 layers performs better than **ReLU** with 30 layers, I am still not sure if this is due to 'stretch pants' approach, which mean 30 layers is a bit over-fitting, hence the accuracy is droped by 0.01 point, we can not confirm this until we try with a bigger size layer.
Judging by Figure 3 and Table 3, that may be the case, as we can see the accuracy dropped by 0.02 while increasing the layer size for all three activation functions, along with the precision for **Non-Tech(0)**. And increasing the size does not matter that much after certain size.

	Non-Tech(0)	Tech(1)	accuracy
Activaton Function: relu, Hidden layer: 10	{'precision': 0.5454545454545454, 'recall': 0....	{'precision': 0.9405204460966543, 'recall': 0....	0.910653
Activaton Function: logistic, Hidden layer: 10	{'precision': 0.0, 'recall': 0.0, 'f1-score': ...	{'precision': 0.9037800687285223, 'recall': 1....	0.90378
Activaton Function: tanh, Hidden layer: 10	{'precision': 0.6, 'recall': 0.642857142857142...	{'precision': 0.9616858237547893, 'recall': 0....	0.924399
Activaton Function: relu, Hidden layer: 20	{'precision': 0.7619047619047619, 'recall': 0....	{'precision': 0.9555555555555556, 'recall': 0....	0.941581
Activaton Function: logistic, Hidden layer: 20	{'precision': 0.0, 'recall': 0.0, 'f1-score': ...	{'precision': 0.9037800687285223, 'recall': 1....	0.90378
Activaton Function: tanh, Hidden layer: 20	{'precision': 0.48148148148148145, 'recall': 0...	{'precision': 0.9431818181818182, 'recall': 0....	0.900344
Activaton Function: relu, Hidden layer: 30	{'precision': 0.6818181818181818, 'recall': 0....	{'precision': 0.9516728624535316, 'recall': 0....	0.931271
Activaton Function: logistic, Hidden layer: 30	{'precision': 0.6363636363636364, 'recall': 0....	{'precision': 0.9479553903345725, 'recall': 0....	0.924399
Activaton Function: tanh, Hidden layer: 30	{'precision': 0.5, 'recall': 0.464285714285714...	{'precision': 0.9433962264150944, 'recall': 0....	0.90378

Table 2: Classification report for hidden layers of $\mathbf{10 \times 10, 20 \times 20, 30 \times 30}$

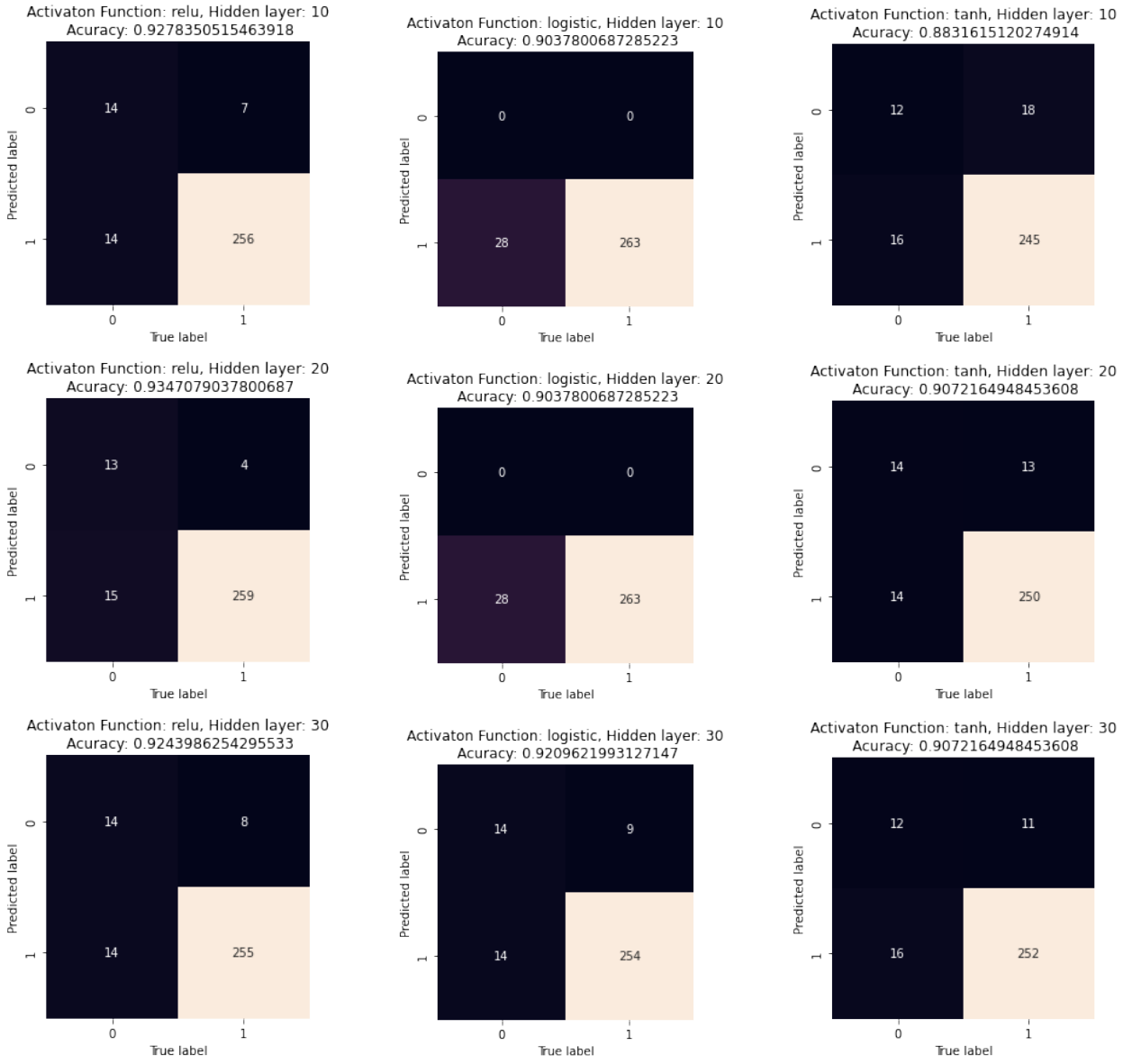

Figure 2: Part 1 Neural Network Hyper-parameters: 3 activation functions of **ReLU, Tanh, Logistic(Sigmoid)**, with 3 sets of hidden layers of $\mathbf{10 \times 10, 20 \times 20, 30 \times 30}$, Hyper-parameters as default in MLPClassifier, such as: batch size=200, learning rate=0.001, max iteration=200,optimization method=adam.

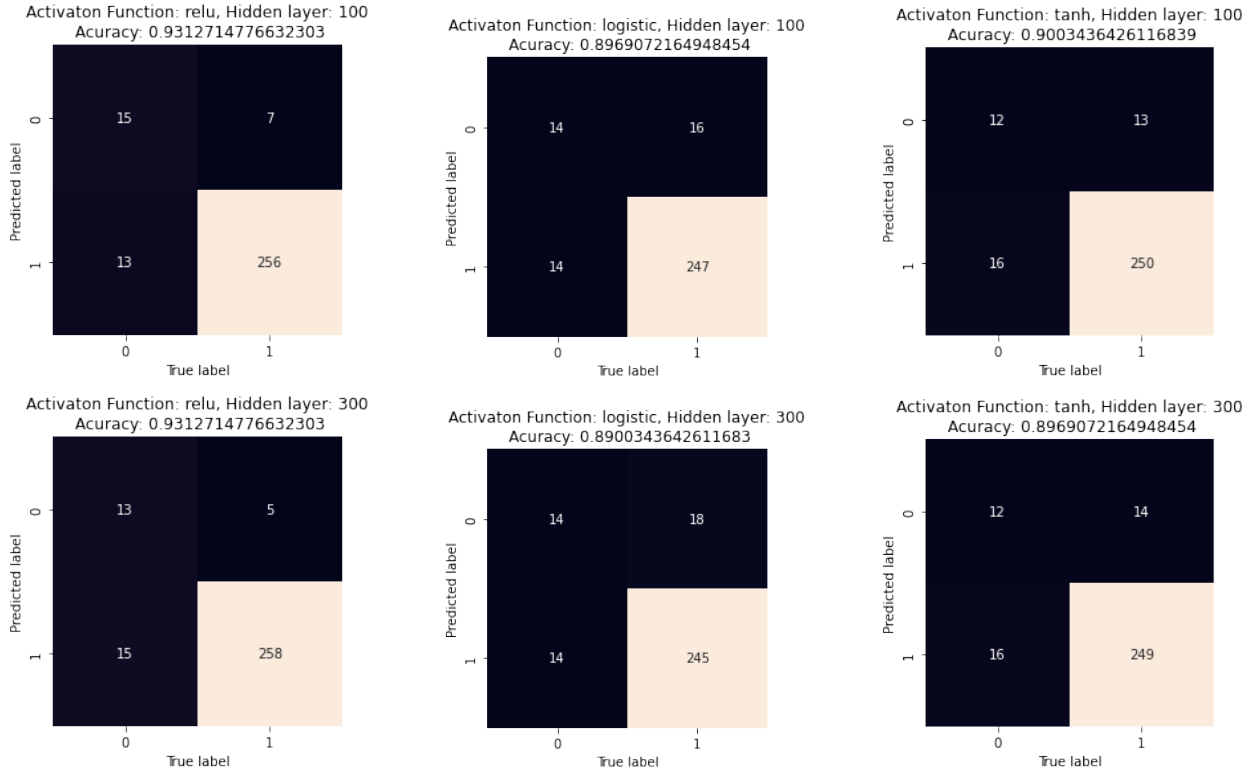

Figure 3: Part 1 Neural Network Hyper-parameters: 3 activation functions of **ReLU, Tanh, Logistic(Sigmoid)**, with 2 sets of hidden layers of $\mathbf{100 \times 100, 300 \times 300}$, Hyper-parameters as default in MLP-Classifier, such as: batch size=200, learning rate=0.001, max iteration=200,optimization method=adam.

	0	1	accuracy
Activaton Function: relu, Hidden layer: 100	{'precision': 0.6818181818181818, 'recall': 0....	{'precision': 0.9516728624535316, 'recall': 0....	0.931271
Activaton Function: logistic, Hidden layer: 100	{'precision': 0.4666666666666667, 'recall': 0....	{'precision': 0.946360153256705, 'recall': 0.9...	0.896907
Activaton Function: tanh, Hidden layer: 100	{'precision': 0.48, 'recall': 0.42857142857142...	{'precision': 0.9398496240601504, 'recall': 0....	0.900344
Activaton Function: relu, Hidden layer: 300	{'precision': 0.7222222222222222, 'recall': 0....	{'precision': 0.945054945054945, 'recall': 0.9...	0.931271
Activaton Function: logistic, Hidden layer: 300	{'precision': 0.4375, 'recall': 0.5, 'f1-score...	{'precision': 0.9459459459459459, 'recall': 0....	0.890034
Activaton Function: tanh, Hidden layer: 300	{'precision': 0.46153846153846156, 'recall': 0...	{'precision': 0.939622641509434, 'recall': 0.9...	0.896907

Table 3: Classification report for hidden layers of $\mathbf{100 \times 100, 300 \times 300}$

2 Impact of Training Duration and Training Data

We use the same data set as part 1 with the same random 70/30 splits and Doc2Vec embedding, while setting rest of the parameters as constants: hidden layer =100, batch size=200, learning rate=0.001, max iteration=200,optimization method=adam, and activation function with **ReLU** and epoch from 10 to 200 with 10 increment each iteration. The learning curves are shown is Figure 4. Before I ran this experiment, I assume that train with $20\%, 40\%, 60\%, 80\%$, and 100% of the training data will gradually be increased by the accuracy rate,yet the results they are pretty much converge to 0.94 to 0.95. What is interesting is the accuracy during the iterations: with 20% stared with the lowest accuracy around 0.60 tp full training set starting with 0.92. My guess is that with lesser the training, it take the model longer time to learning from, but eventually improving the accuracy with each epoch, this again confirms the textbook's claim that the more data the better, along with bigger iteration.
We can clearly see the influence of bigger the data percentage selected for the training the better accuracy showing in the early stage. The question of influence of the training duration not only depend on epoch size but also the maximum iteration setting. As shown in Figure 5, after 200 epoch, the accuracy stay constant with a high starting point as we set the starting epoch as 50 instead of 10.

Additional notes

The **Doc2Vec** is already a matured supervised learning algorithm, so it helped this lab with majority of the effort since the embedding of the text info already been 'learned', by resenting dense matrix. The data is skewed toward **Tech(1)**, so once we collect enough **Non-Tech(0)** into our training, or even tune down the volume of the Tech set, the precision for **Non-Tech(0)** substantially, in my opinion.

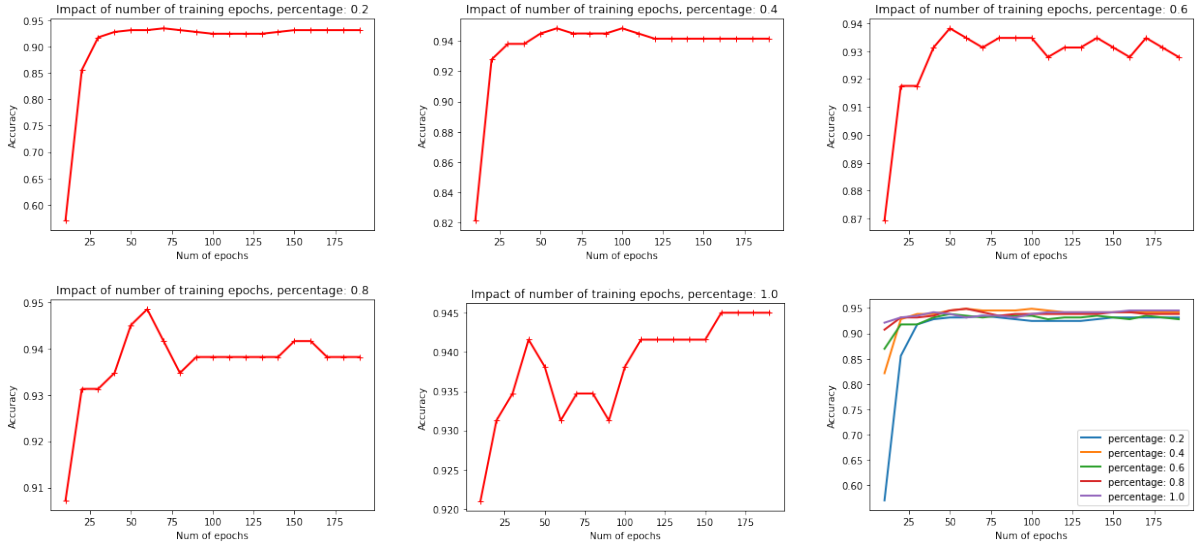

Figure 4: Five learning curves

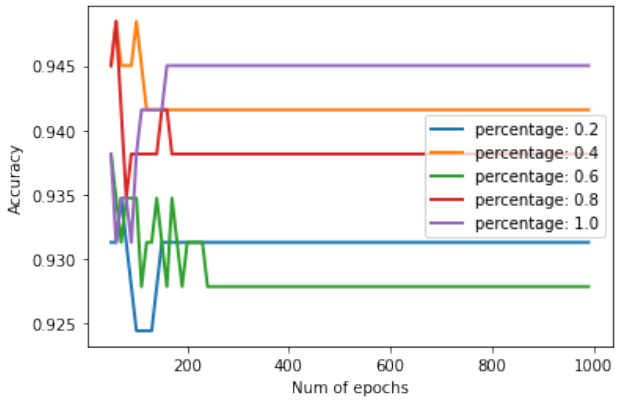

Figure 5: 50 to 500 epoch

Code

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler


def create_categories(x):
    '''
    change tech column to 1,
    non-tech to 0
    '''
    if x == "Tech":
        return 1
    else:
        return 0


def add_embedding(data):
    '''
    adding doc2vec column based on text
    '''
    # tokenize and tag the card text
    card_docs = [TaggedDocument(doc.split(' '), [i])
                 for i, doc in enumerate(data.text)]

    # instantiate model
    model = Doc2Vec(vector_size=64, window=2,
                    min_count=1, workers=8, epochs=40)
    # build vocab
    model.build_vocab(card_docs)
    # train model
    model.train(card_docs, total_examples=model.corpus_count,
                epochs=model.epochs)
    card2vec = [model.infer_vector((data['text'][i].split(' ')))
                for i in range(0, len(data['text']))]
    # Create a list of lists
    dtv = card2vec
    # set list to dataframe column
    data['doc2vec'] = dtv
    return data


def split_data(test_ratio=None):
    '''
    split the data with
    test_radio input: .3
    '''
    return train_test_split(X, y, test_size=test_ratio, random_state=1)


def train_with_activation_funciton_nnodes(activation_function=None, n_hidden_nodes=None):
    # Standardize data
    stdsc = StandardScaler()
    stdsc.fit(X_train)
    X_train_std = stdsc.transform(X_train)
    X_test_std = stdsc.transform(X_test)
    # activation{'identity', 'logistic', 'tanh', 'relu'}, default='relu'
    mlp = MLPClassifier(activation=activation_function, hidden_layer_sizes=[
                        n_hidden_nodes, n_hidden_nodes])

    mlp.fit(X_train_std, y_train)
```

```
    y_predicted = mlp.predict(X_test_std)
    # other metric
    report = metrics.classification_report(
        y_test, y_predicted, output_dict=True)
    # plot
    mat = metrics.confusion_matrix(y_test, y_predicted)
    sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
    plt.xlabel("True label")
    plt.ylabel("Predicted label")
    plt.title("Activaton Function: {}, Hidden layer: {}\n Acuracy: {}".format(
        activation_function, n_hidden_nodes, mlp.score(X_test_std, y_test)))
    plt.show()

    return report


def training_by_percentage(percentage=None):
    start_num_epochs = 10
    finish_num_epochs = 200
    inc_amt = 10

    pred_scores = []
    num_epochs = []
    # standard then slice
    X_train_slice = X_train[:round(percentage*len(X_train))]
    X_train_std_slice = stdsc.transform(X_train_slice)
    X_test_std = stdsc.transform(X_test)
    y_train_slice = y_train[:round(percentage*len(y_train))]

    for epoch_count in range(start_num_epochs, finish_num_epochs, inc_amt):
        my_classifier = MLPClassifier(
            activation='relu', random_state=20, max_iter=epoch_count)
        my_classifier.fit(X_train_std_slice, y_train_slice)
        score = my_classifier.score(X_test_std, y_test)
        pred_scores.append(score)
        num_epochs.append(epoch_count)

#     plt.figure()
    plt.plot(num_epochs, pred_scores, 'r-+', linewidth=2)
    plt.xlabel("Num of epochs")
    plt.ylabel("Accuracy")
    plt.title(
        "Impact of number of training epochs, percentage: {}".format(percentage))
    plt.show()


if __name__ == "__main__":

    tech_data = pd.read_csv('tech_nontech_classification_annotations.csv')
    tech_data = add_embedding(tech_data)
    tech_data["binary"] = tech_data['accept'].apply(create_categories)
    X = (np.vstack(np.array(tech_data['doc2vec'])))
    y = np.array(tech_data['binary'])
    X_train, X_test, y_train, y_test = split_data(test_ratio=.3)
    # part1:
    reports = {}
    for hiden in [10, 20, 30]:
        for act_func in ['relu', 'logistic', 'tanh']:
            report = train_with_activation_funciton_nnodes(
                activation_function=act_func, n_hidden_nodes=hiden)
            reports["Activaton Function: {}, Hidden layer: {}"
                    .format(act_func, hiden)] = report
    # print(pd.DataFrame(reports).T[['0','1','accuracy']].to_latex())
    # part2:
    percent_slice = [.2, .4, .6, .8, 1.0]
    for percentage in percent_slice:
        training_by_percentage(percentage=percentage)
```