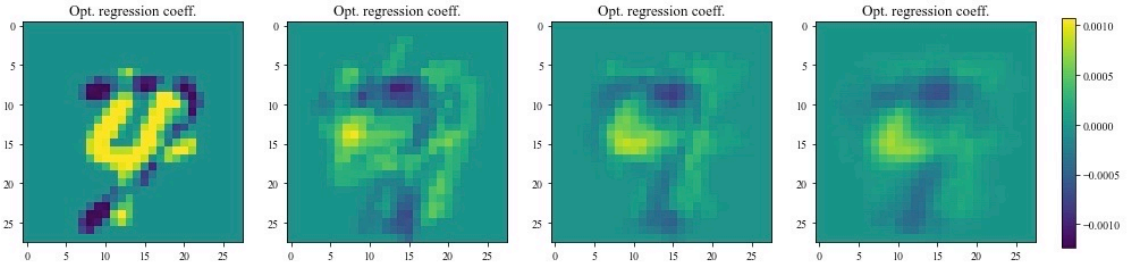
Excercise 1

i)

MNIST Binary Classification by MNB for 4 vs. 7



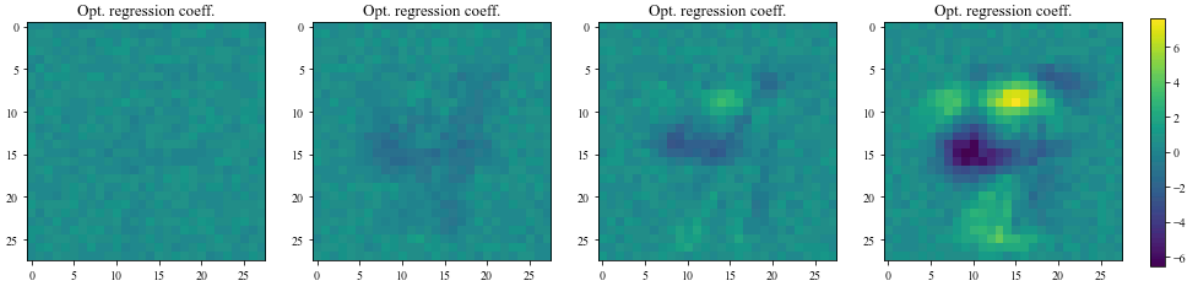
Accuracy = 0.609
Sensitivity = 0.997
Specificity = 0.23
Precision = 0.988
Fall_out = 0.003
Miss_rate = 0.77
train size = 2

Accuracy = 0.929
Sensitivity = 0.915
Specificity = 0.942
Precision = 0.919
Fall_out = 0.085
Miss_rate = 0.058
train size = 10

Accuracy = 0.947
Sensitivity = 0.984
Specificity = 0.911
Precision = 0.983
Fall_out = 0.016
Miss_rate = 0.089
train size = 30

Accuracy = 0.957
Sensitivity = 0.98
Specificity = 0.934
Precision = 0.98
Fall_out = 0.02
Miss_rate = 0.066
train size = 100

MNIST Binary Classification by LR for 4 vs. 7



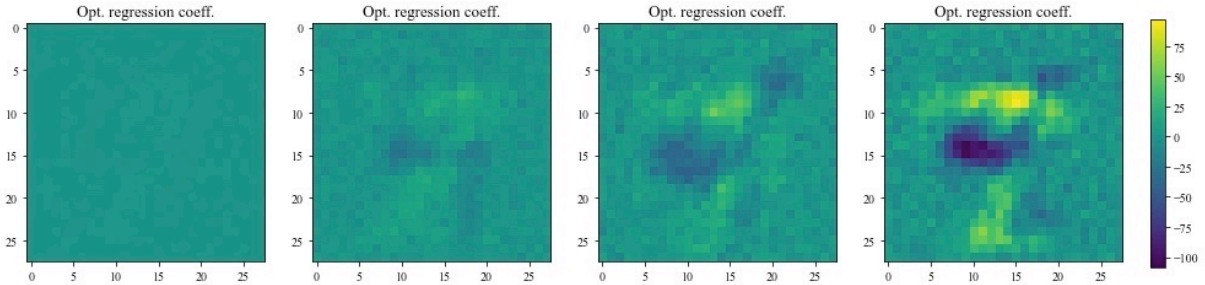
Accuracy = 0.512
Sensitivity = 0.0
Specificity = 1.0
Precision = 0.512
Fall_out = 1.0
Miss_rate = 0.0
train size = 2

Accuracy = 0.505
Sensitivity = 0.998
Specificity = 0.036
Precision = 0.946
Fall_out = 0.002
Miss_rate = 0.964
train size = 10

Accuracy = 0.909
Sensitivity = 0.816
Specificity = 0.997
Precision = 0.851
Fall_out = 0.184
Miss_rate = 0.003
train size = 30

Accuracy = 0.967
Sensitivity = 0.947
Specificity = 0.987
Precision = 0.951
Fall_out = 0.053
Miss_rate = 0.013
train size = 100

MNIST Binary Classification by PR for 4 vs. 7



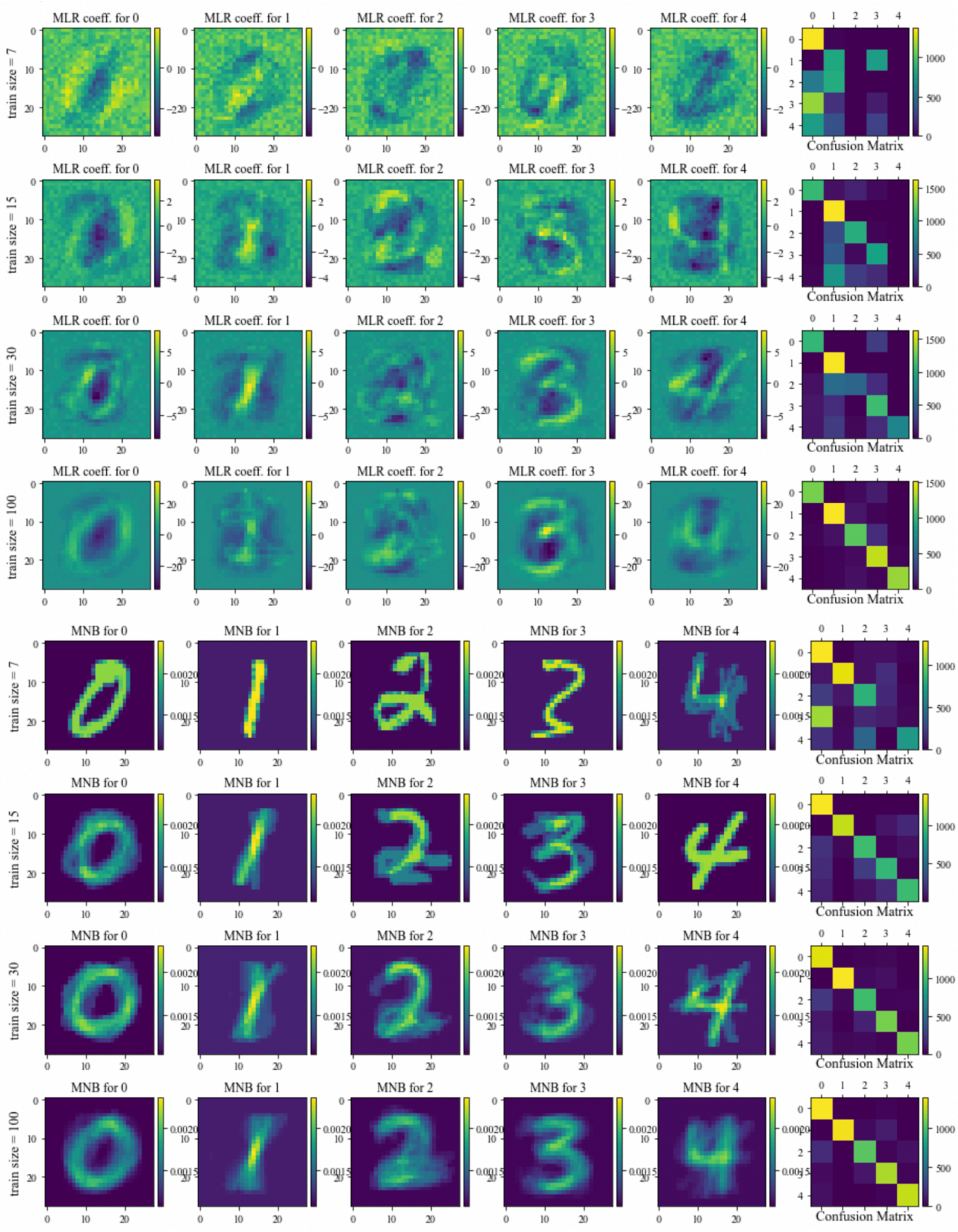
Accuracy = 0.525
Sensitivity = 0.018
Specificity = 1.0
Precision = 0.521
Fall_out = 0.982
Miss_rate = 0.0
train size = 2

Accuracy = 0.713
Sensitivity = 0.421
Specificity = 0.987
Precision = 0.645
Fall_out = 0.579
Miss_rate = 0.013
train size = 10

Accuracy = 0.915
Sensitivity = 0.848
Specificity = 0.978
Precision = 0.873
Fall_out = 0.152
Miss_rate = 0.022
train size = 30

Accuracy = 0.971
Sensitivity = 0.976
Specificity = 0.966
Precision = 0.977
Fall_out = 0.024
Miss_rate = 0.034
train size = 100

ii)



For some reason my code would not work w/ small training sets so I trained on [7, 15, 30, 100]. We can see that w/ Bayes method our model did much better w/ small training sizes such as 7, 15 and even 30. It seems to be that Bayes model which assumes features are independent actually helps our model. Even with the larger training size, our confusion matrix looks better in Bayes version.

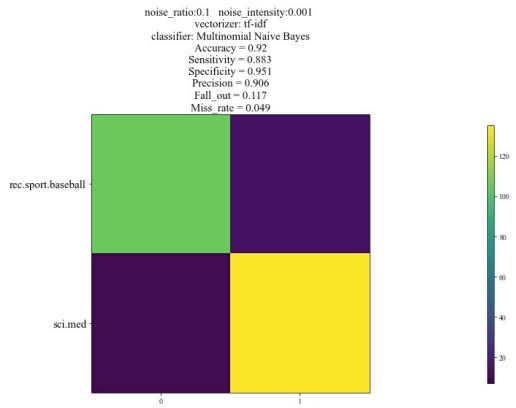
Exercise 2

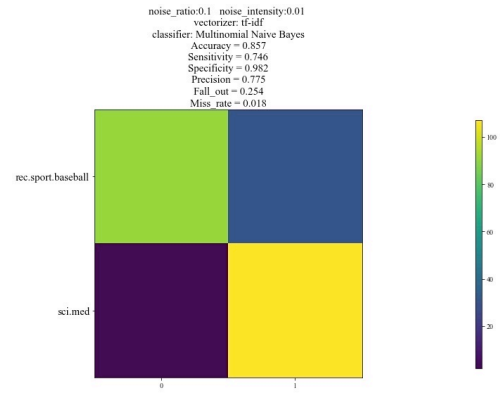
i)

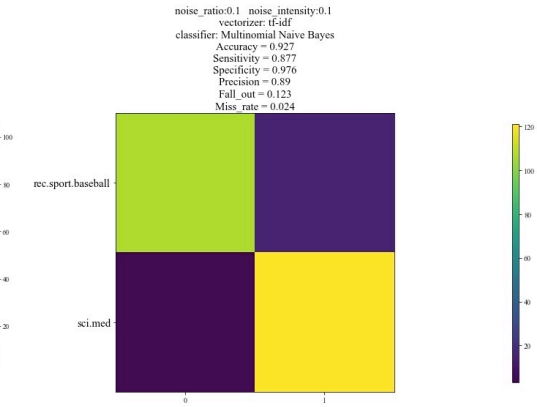
```
def sample_multiclass_20NEWS(list_classes=[0, 1], full_data=None, vectorizer = 'tf-idf', verbose=True, noise_ratio=0, noise_intensity=1):
    # get train and test set from 20NewsGroups of given categories
    # vectorizer \in ['tf-idf', 'bag-of-words']
    # documents are loaded up from the following 10 categories
    categories = [
     'comp.graphics',
     'comp.sys.mac.hardware',
     'misc.forsale',
     'rec.motorcycles',
     'rec.sport.baseball',
     'sci.med',
     'sci.space',
     'talk.politics.guns',
     'talk.politics.mideast',
     'talk.religion.misc'
     ]

    data_dict = {}
    data_dict.update({'categories': categories})

    if full_data is None:
        remove = ('headers','footers','quotes')
        stopwords_list = stopwords.words('english')
        stopwords_list.extend(['thanks','edu','also','would','one','could','please','really','many','anyone','good','right','get','even','want','m

        newsgroups_train_full = fetch_20newsgroups(subset='train', categories=categories, remove=remove) # raw documents
        newsgroups_train = [re.sub(r'\d+','', file) for file in newsgroups_train_full.data] # remove numbers (we are only interested in words)
        y = newsgroups_train_full.target # document class labels
        Y = list2onehot(y.tolist(), list_classes)


        if vectorizer == 'tfidf':
            vectorizer = TfidfVectorizer(stop_words=stopwords_list)
        else:
            vectorizer = CountVectorizer(stop_words=stopwords_list)

        X = vectorizer.fit_transform(newsgroups_train) # words x docs  # in the form of sparse matrix
        X = np.asarray(X.todense())
        print('!! X.shape', X.shape)
        idx2word = np.array(vectorizer.get_feature_names())   # list of words that corresponds to feature coordinates

        data_dict.update({'newsgroups_train': data_cleaned})
        data_dict.update({'newsgroups_labels': y})
        data_dict.update({'feature_matrix': vectors})
        data_dict.update({'idx2word': idx2word})

    else:
        X, y = full_data

    idx = [i for i in np.arange(len(y)) if y[i] in list_classes] # list of indices where the label y is in list_classes
```

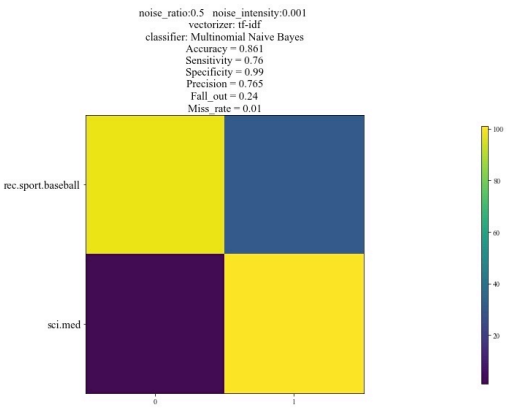
```
X01 = X[idx,:]
Y01 = Y[idx,:]

X_train = []
X_test = []
y_test = [] # list of one-hot encodings (indicator vectors) of each label
y_train = [] # list of one-hot encodings (indicator vectors) of each label

for i in np.arange(X01.shape[0]):
    # for each example i, make it into train set with probabiliy 0.8 and into test set otherwise
    U = np.random.rand() # Uniform([0,1]) variable
    if U<0.8:
        if noise_ratio != 0:
            for j in np.arange(X01.shape[1]):
                R = np.random.rand()
                V = np.random.rand()
                if R < noise_ratio:
                        X01[i,j] += V*noise_intensity
        X_train.append(X01[i,:])
        y_train.append(Y01[i,:].copy())
    else:
        X_test.append(X01[i,:])
        y_test.append(Y01[i,:].copy())

X_train = np.asarray(X_train)
X_test = np.asarray(X_test)
y_train = np.asarray(y_train)
y_test = np.asarray(y_test)

data_dict.update({'X_train': X_train})
data_dict.update({'X_test': X_test})
data_dict.update({'y_train': y_train})
data_dict.update({'y_test': y_test})

return X_train, X_test, y_train, y_test, data_dict
```

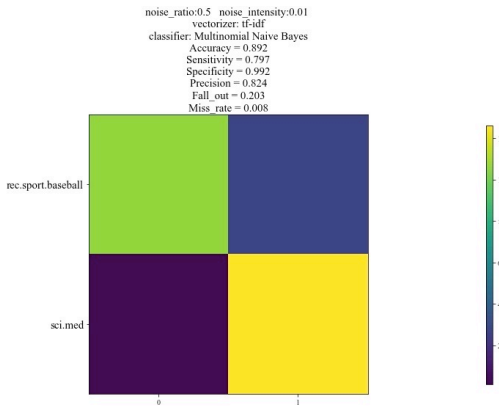
ii)

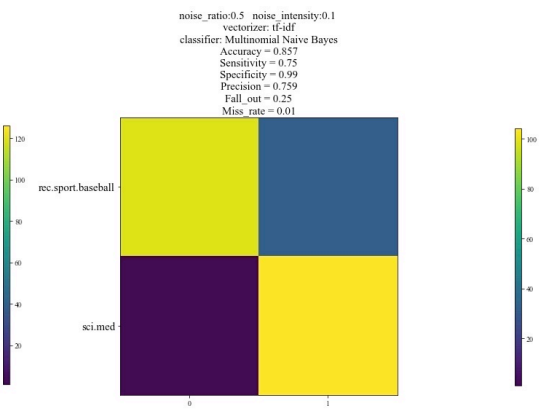


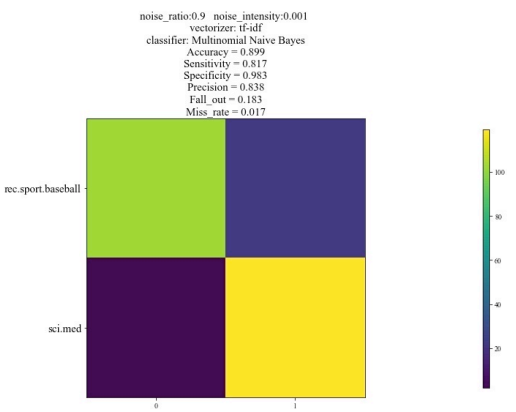


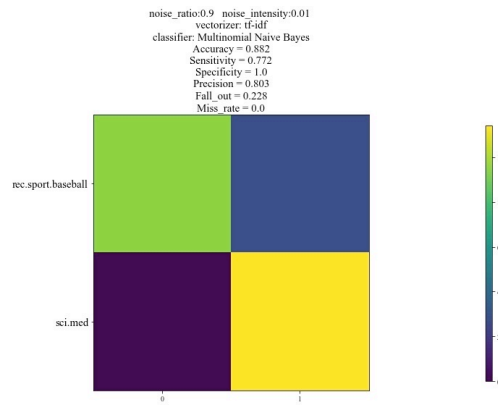


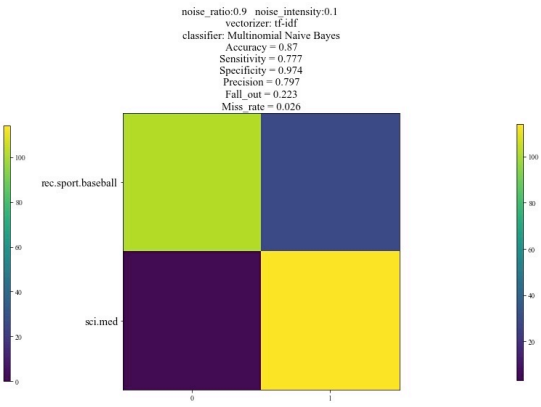






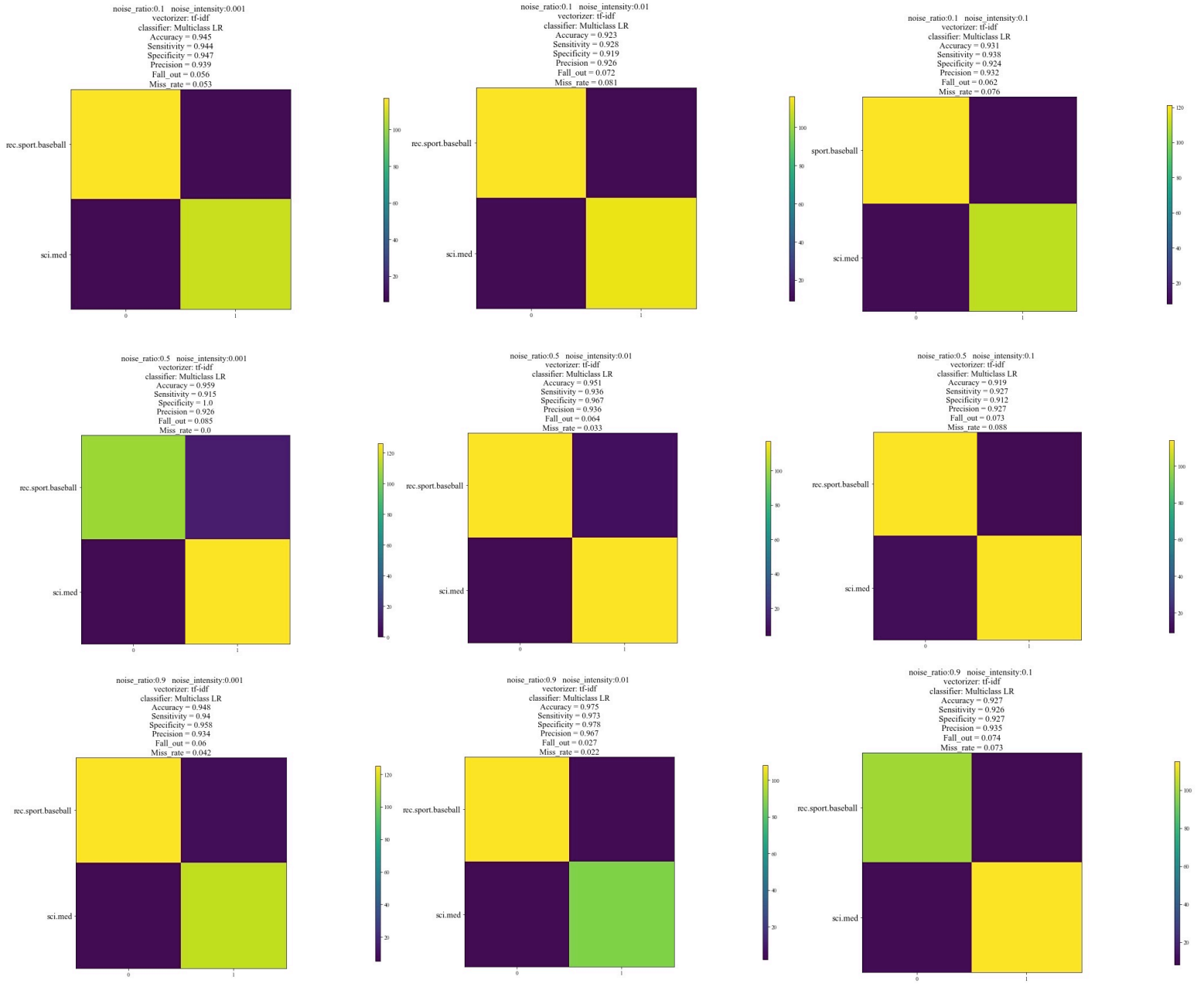






For the most part, performance remains strong throughout. We can see that there is a small performance dip as we go south-east. (Increase noise_ratio and increase noise_intensity). The dip in performance is so small (assuming performance=accuracy) that we can say it's negligible. The most noticeable change in our metric as we go south-east is the fall of precision. The top left metric has precision of 90%, while at the bottom right we see precision fall to 79%. Bayes model assumes our features do NOT have a relationship with each other, this may be the reason our model performs well since noisy words don't affect others.

iii)



In this experiment, it looks like linear regression performs much better than the naive bayes model. Here though we see a drop in accuracy as we move right (increasing the noise_intensity). There doesn't seem to be a affect in the noise_ratio variable change, but we can see that noise_intensity on the other hand plays a bigger role on the dip of accuracy. It seems to be that any amount of intense noise can affect performance, unlike Bayes experiment where neither noise_ratio or noise_intensity individually played a role in affecting performance.

Exercise 3

i)

Now remember from exercise that we have a constraint s.t. $\sum q_i = 1$. we can now use lagrange multiplier to solve.

$$\frac{\partial}{\partial q_i}\left[\sum_{i=1}^{k}\sum_{s=1}^{N} 1(y_s=1)\log q_i - \lambda\left(\sum_i q_i - 1\right)\right]$$

$$= \sum_{s=1}^{N} 1(y_s=1)\cdot\frac{1}{q_i} - \lambda$$

Now setting equal to zero:

$$\frac{\sum_{s=1}^{N} 1(y_s=1)}{q_i} = \lambda \Rightarrow \frac{\sum_{s=1}^{N} 1(y_s=1)}{\lambda} = q_i \quad (I)$$

and plugging in our constraint:

$$\frac{\sum_i \sum_{s=1}^{N} 1(y_s=1)}{\lambda} = \sum_i q_i = 1$$

$$\lambda = \sum_{i,s}^{N} 1(y_s=1)$$

and now we can plug in back into (I):

$$\hat{q}_i = \frac{}{\sum_{i,s} 1(y_s=1)}$$

ii) $$\ell_{GNB}(w) := \rho \sum_{i=1}^{u}\left(\sum_{s=1}^{N} 1(y_s=i)\right)\log q_i - \frac{1}{2}\sum_{j=1}^{p}\sum_{i=1}\left(\sum_{s=1}^{N} 1(y_s=i)\log(2\pi\sigma_{ij}^2)\right.$$
$$\left. - \sum_{s=1}^{N}\sum_{j=1}^{p}\sum_{i=1}^{u} 1(y_s=i)\frac{(\phi_i(x_s)-\mu_{ij})^2}{2\sigma_{ij}^2}\right.$$

$$\frac{2\,\ell_{GNB}(w)}{2\mu_{ij}} = + \sum_{s=1}^{N} \frac{1(y_s=i)\cdot 2(\phi_j(x_s)-\mu_{ij})\cdot -1}{2\sigma_{ij}^2}$$

$$= \sum_{s=1}^{N} 1(y_s=i)\frac{(\phi_j(x_s)-\mu_{ij})}{\sigma_{ij}^2}$$

Now setting it equal to 0:

$$\sum_{s=1}^{n} 1(y_s = i)\frac{(\phi_j(x_s) - \mu_{ij})}{\sigma_{ij}^2} = 0$$

$$\frac{\sum_{s=1}^{n} 1(y_s = i)\,\phi_j(x_s)}{\sigma_{ij}^2} = \frac{\sum_{s=1}^{n} 1(y_s = i)\,\mu_{ij}}{\sigma_{ij}^2}$$

$$\frac{\sum_{s=1}^{n} 1(y_s = i)\,\phi_j(x_s)}{\sum_{s=1}^{n} 1(y_s = i)} = \mu_{ij}$$

$$\downarrow$$

$$\hat{\mu}_{ij} = \frac{\sum_{s=1}^{n} 1(y_s = i)\,\phi_j(x_s)}{\sum_{s=1}^{n} 1(y_s = i)}$$

← sum of feature ϕ_j when entry is of class i

← total entries of class i

j = feature
i = class

so we can see above that we have the sum of feature result with class i divided by total sum of class i which gives us an average of feature j within class i.

iii) $$\ell_{GNB}(w) := \rho \sum_{i=1}^{n}\left(\sum_{s=1}^{n} 1(y_s = i)\right)\log q_i - \frac{1}{2}\sum_{j=1}^{p}\sum_{i=1}^{n}\left(\sum_{s=1}^{n} 1(y_s = i)\log(2\pi\sigma_{ij}^2)\right.$$
$$\left. - \sum_{s=1}^{n}\sum_{j=1}^{p}\sum_{i=1}^{n} 1(y_s = i)\frac{(\phi_j(x_s) - \mu_{ij})^2}{2\sigma_{ij}^2}\right.$$

$$\frac{2\ell_{GNB}(w)}{\sigma_{ij}} = -\frac{1}{2}\sum_{s=1}^{n} 1(y_s = i)\cdot\frac{4\pi\sigma_{ij}}{2\pi\sigma_{ij}^2} + \frac{\sum_{s=1}^{n} 1(y_s = i)(\phi_j(x_s) - \mu_{ij})^2}{\sigma_{ij}^{-3}}$$

$$= -\frac{\sum_{s=1}^{n} 1(y_s = i)}{\sigma_{ij}} + \frac{\sum_{s=1}^{n} 1(y_s = i)(\phi_j(x_s) - \mu_{ij})^2}{\sigma_{ij}^{-3}}$$

$$= -\sigma_{ij}^{-1}\left(\sum_{s=1}^{N} 1(y_s=i)\right) + \sigma_{ij}^{-3}\sum_{s=1}^{N} 1(y_s=i)(\phi_j(x_s)-\mu_{ij})^2$$

Now setting it equal to 0:

$$-\sigma_{ij}^{-1}\left(\sum_{s=1}^{N} 1(y_s=i)\right) + \sigma_{ij}^{-3}\sum_{s=1}^{N} 1(y_s=i)(\phi_j(x_s)-\mu_{ij})^2 = 0$$

$$-\sigma_{ij}^{3}\cdot -\sigma_{ij}^{-1}\left(\sum_{s=1}^{N} 1(y_s=i)\right) = -\sigma_{ij}^{-3}\sum_{s=1}^{N} 1(y_s=i)(\phi_j(x_s)-\mu_{ij})^2 \cdot -\sigma_{ij}^{3}$$

$$\sigma_{ij}^2 = \frac{\sum_{s=1}^{N} 1(y_s=i)(\phi_j(x_s)-\mu_{ij})^2}{\sum_{s=1}^{N} 1(y_s=i)}$$

feature j in class i minus avg of feature j within class i

total entries of class i

and thus we can see above that we have the sum of varences over our total entries of class i giving us the varience of class-i emperical distribution!

Exercise 4

$$L_{total}(w) = -\log L(y_1, \dots, y_n; w)$$

assuming output is generated by a k-dimensional multivariate normal distribution $N(\hat{y}(x_i; w), \sigma^2 I)$ & I is $k \times k$ identity matrix

$$= -\log \prod_s \frac{1}{(2\pi\sigma^2)^{1/2}} e^{\frac{-(y_s - \hat{y}_s)^2}{2\sigma^2}}$$

$$= -\sum_s -\frac{1}{2}\log(2\pi\sigma^2) - \frac{(y_s - \hat{y}_s)^2}{2\sigma^2}$$

↓ square loss, doesn't depend on σ

minimize $\hat{y}_s \Rightarrow \frac{1}{2}\sum_s (y_s - \hat{y}_s)^2$

which then leads us to our desired derivation of

$$\ell(y_s, \hat{y}(x_s; w)) = \frac{1}{2}\|y_s - \hat{y}(x_s; w)\|_F^2$$

where then

$$w := \operatorname{argmin}_w \left[L_{NN}(w) := \sum_{s=1}^{n} \ell(y_s, \hat{y}(x_s; w)) \right]$$

■