Proceedings of Machine Learning Research 1–5

A Mini-Replication of CNM06

John Cooper JLCOOPER@UCSD.EDU

Abstract

Drawing from the CNM06 paper, I attempt to perform a mini-scale replication of its methods and results, using only three algorithms and five scoring metrics (some of which are not used in the original paper). I compare KNN, SVM, and Logistic Regression.

Keywords: *CNM06* - Caruana Paper, *COV* - Cover Type Data, *Letter OPos* - Letter Data "O" positive classification, *Letter AMPos* - Letter Data "AM" positive classification

1. Introduction

This project presents a small-scale analysis of a select subset of data sets, algorithms, and performance metrics found CNM06. I attempt to compare the binary-classification performances of SVM, KNN, and Logit across four data sets and five scoring metrics.

By undertaking a small-scale analysis (voluntarily) constrained by four data sets and classifiers with the sole intention of comparing performance, bluntly using a single performance metric alone (accuracy) did not seem like an appropriate method to capture the differences between these classifiers. As two of the four data sets analyzed were quite imbalanced (ADULT, LetterOPos), more metrics were needed to compensate for this, including Precision, Recall, F1, and ROC-AUC.

To address certain classifiers, such as SVM, which are not meant to predict probabilities (Caruana, 2006), I did **not** calibrate with Platt Scaling or Isotonic Regression as CNM06 does. Instead, the decision outputs were mapped to probabilities through an **untrained** logit function. The pitfalls of this post-scaling are not addressed, and so roc-performance comparisons between SVM and the other classifiers should be taken hesitantly.

The results of this analysis were surprisingly consistent with the CNM06 paper. Table 2 (Mini-Replication) shows the performances over metrics for each algorithm, where SVM scores slightly higher than KNN and significantly higher than LOGIT under all metrics. This is consistent with the Table 2 in the CNM06 paper (6-6). The difference in per-metric-scoring means between the SVM and the other classifiers were also found to be statistically significant.

By problem, SVM outscored KNN on all data sets and LOGIT on all but one. Interestingly, the scores of KNN on LetterOPos and LetterAMPos were statistically indistinguishable from greater SVM scores on those data sets. This is relatively consistent with CNM06, wherein Table 3 (CNM06) KNN actually outperforms SVM on few metrics, although not by much (6-6).

2. Methods

For the algorithms selected, their hyperparameter selection follows those presented in CNM06 almost verbatim, while the performance metrics slightly differ.

2.1. Classifiers

SVMs: Kernels used were radial basis function, linear, and polynomial kernels. For the rbf, gammas used were $\{.001, .005, .01, .05, .1, .5, 1, 2\}$, and for the polynomial kernel, degrees $\{2, 3\}$ were used. For each of these, their regularization parameter varied from 10^{-7} to 10^{4}.

KNN: Hyperparameters varied were the number of neighbors and the types of weights. Number of neighbors took equally spaced (20-step) values between 1 and 500 (exclusive). Weights searched were $\{uniform, distance\}$.

Logit: This classifier was initialized with solver set to Limited-memory BFGS and max iterations set to 1000, the latter to encourage convergence. The penalization term took values in $\{l2, none\}$ and the regularization term ranged from 10^{-8} to 10^{5}.

2.2. Metrics

The performance metrics used in this analysis are Accuracy, ROC, Precision, Recall, and F1.

Accuracy is used as the base measure for comparison across algorithms and data sets, although as mentioned in Section 1 this metric fails to capture to the true performance of algorithms on imbalanced data sets. Even a poor classifier will produce high accuracy on a data set with extreme class imbalance, since by simply guessing at the over-represented classification, the proportion of correct guesses should roughly parallel the over-represented class. For this reason, in negative-class imbalanced data sets, we turn to ROC, Precision, Recall, and F1.

ROC and Precision operate irrespective of class distribution, and measure the efficiency with which classifiers order positive cases before negative cases (Caruana, 2006). Precision provides information on how well our classifier is able to correctly predict positive cases from the pool of all existing positive labels. Recall, in the case of negative imbalance, will tend to be small if the classifier is leans towards negative classification, as the trade-off between true positive and false negative will favor the latter. F1 is used to incorporate both these ideas. All four will provide us a measure of how poorly our classifier is actually doing despite high accuracy.

2.3. Comparisons Across Metrics

Unlike CNM06, calibration is not used due to (1) the computational load of five-fold grid searches nested in multiple three-fold cross-validated calibrations (Platt/Iso), and (2) time constraints. Both KNN and Logit are equipped with simple probability mappings, whereas SVM outputs decisions which *can* be mapped to probabilities through a sigmoid function (Platt)(6-1)

$$P(y = 1|x) = \frac{1}{1 + e^{Af(x)+B}}$$

whose parameters $\{A, B\}$ are tuned through a k-fold cross validation on the training set. As mentioned in Section 1, this procedure was not followed. The decisions of the SVM were simply mapped to probabilities through

$$\frac{e^x}{1 + e^x},$$

which is clearly not best practice, especially since we are training highly non-linear SVMs. Aside from SVM performance, performance on other metrics and algorithms are not calibrated. Instead,

their probability prediction functions are used, which internally seem to use a sigmoid for mapping (6-3).

2.4. Data

The four data sets used were LETTER1(OPos), LETTER2 (AMPos), COVTYPE, and ADULT (6-7). For each data set, all categorical features were converted using one-hot-encoding.

For the ADULT data set, the target variable (income) was converted to binary classification through assigning the mapping $(> 50k) \rightarrow 1$ and $(<= 50k) \rightarrow 0$.

For COV, the target variable ("cov") was converted to binary by assigning the largest class to 1 and the rest to 0.

For LETTER, the target variable ("lettr") was split into two cases as in CNM06. First, "O" was mapped to 1 and the rest of the letters were mapped to 0. Next, "A-M" was mapped to 1, and the rest of the letters were mapped to 0. Classification was done on both targets.

3. Training and Testing Framework

First a data set is chosen and randomly sampled from three times, each time breaking the data into a training set (5000) and a testing set (# samples - 5000).

A classifier is then selected.For each of train/test splits, a five-fold-cv gridsearch with this classifier is run to obtain the best parameters. The classifier with the best parameters is then retrained on the entire train set and used to make predictions on the test set, yielding a test score.

This train/test process is repeated on the two remaining train-test pairs, yielding two more test scores.

From these we obtain three test scores for a single classifier over a single data set. This entire process is repeated for every classifier-dataset pair. Thus, under a performance single metric, one classifier will have a total of **12** test scores to report as well as **12** validation scores.

Table 1 lists the set-up of each problem: the data set, the number of features, the train size, test size, and train size percentage of total data

Note:
The actual implementation differs in structure from what is described above, but only superficially. The implementation runs a for loop to simulate trials, and within each loop splits the data with a random seed corresponding to the trial number before the classifier is trained. The random seed acts to preserve consistency in splitting (as if the actual implementation followed the above).

4. Performance

Tables are presented which summarize performances.

In Table 2, the rows list the three classifiers, while the columns list the metrics. For each classifier-data combo, three scoring performances are produced. This produces nine scoring performances (each performance encapsulating the five metrics used) in total for each classifier. The entries in this table represent the *averaged* scores per classifier, by metric. Boldfaced scores represent a score (under a specific metric and algorithm) whose population mean has been determined to be larger than the others under a 95% confidence process.

In Table 3, the rows list the three classifiers used, while the columns list the data sets. The entries represent the averages of each classifier across all five metrics on a

single data set. Since there are three trials, each metric corresponds to three performance instances over those three trials. Given that there are five metrics, we sum over the five metrics entry-wise (by trial) adn divide by the number of metrics to obtain an average over the five metrics per trial. After this we sum these averages and divide by the number of trials.

Note:
Within the testing/training framework, the grid search selects the optimal classifier based on the classifier's internal scoring method - for SVM, Logit, and KNN, this scoring method is ACC. So when we tune our parameters, we are optimizing based on this metric. Thus when we compare classifiers with metrics other than ACC, we are actually comparing how well classifiers, which are optimized on accuracy, perform under criterion not inherent to the optimization process. This is why there are no validation errors for metrics other than mean accuracy in Table 4.

5. Discussion-Conclusion

5.1. Test Performance Summary

The best classifier on the LetterOPos and LetterAMPos data sets is SVM, whereas on the COV and ADULT data sets, SVM and LOGIT show the best performances, respectively. LOGIT surprisingly outperforms SVM and KNN on the ADULT data set. In the case of LOGIT on the ADULT data set, there is no other score that rivals it with statistical significance, whereas in the case of LetterOPos and LetterAMPos, SVM and KNN perform at statistically indistinguishable levels (Table 3, Table 6).

Over scoring metrics, SVM uniformly outperforms the other classifiers, with LOGIT performing uniformly worse across all metrics against the other two classifiers. While this seems to show that SVM performs better on all metrics, by problem, we see that some uniformly-best classifier does not exist (Table 2, Table 5).

5.2. Validation Performance

One concerning aspect of this analysis was that validation performance and test performance seemed to run in parallel - sometimes they were equal, other times test performance actually exceeded validation performance (Table 2, Table 4). This is referring specifically to accuracy, as those are the only validation metrics which were saved. In the case of testing scoring on non-accuracy metrics outperforming validation scores computed internally with an accuracy, this shouldn't be too much of a concern. Since the classifiers are optimized on accuracy, it makes that their testing performance under different metrics would not naturally produce a score bounded by accuracy validation scores.

5.3. Non-accuracy metrics

It was necessary to include other metrics to gauge classifier performance, especially on the imbalanced data sets COV and ADULTOPos. Table 3 shows the difference between performance metrics, with ROC and ACC showing the best performances respectively, followed by REC, PREC, and F1, not exactly in that order. The introduction of these metrics was the only way for us to see how poorly LOGIT actually performs relative to the other classifiers on most data sets. Furthermore, accuracy would have generally inflated our view of all classifiers performances, especially on imbalanced data (Table 2), in some cases widening

and other cases shrinking the performance margins.

5.4. Calibration

As mentioned, calibration was not used in this analysis. Without a uniform mapping of prediction (or decision functions) to probabilities, especially between SVM and the other classifiers, the ROC score comparison between SVM and the other classifiers is most likely flawed.

6. Citations and Bibliography

1. "Platt Scaling" Wikipedia, Wikimedia Foundation, 6 Dec. 2020, en.wikipedia.org/wiki/Platt_scaling
2. Scikit-learn.org. 2020. Sklearn.Neighbors.Kneighborsclassifier — Scikit-Learn 0.23.2 Documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html [Accessed 15 December 2020].
3. GitHub. 2020. Scikit-Learn/Scikit-Learn. [online] Available at: https://github.com/scikit-learn/scikit-learn/blob/95d4f0841/sklearn/neural_network/_multilayer_perceptron.py [Accessed 15 December 2020].
4. Scikit-learn.org. 2020. Sklearn.Svm.SVC — Scikit-Learn 0.23.2 Documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html [Accessed 15 December 2020].
5. Scikit-learn.org. 2020. Sklearn.LinearModel.Logisticregression— Scikit-Learn 0.23.2 Documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html[Accessed 15 December 2020].
6. Caruana, R. and Niculescu-Mizil, A., 2020. An Empirical Comparison Of Supervised Learning Algorithms. [online] Cs.cornell.edu. Available at: https://www.cs.cornell.edu/~caruana/ctp/ct.papers/caruana.icml06.pdf
7. Archive.ics.uci.edu. 2020. UCI Machine Learning Repository. [online] Available at: https://archive.ics.uci.edu/ml/index.php [Accessed 15 December 2020].

Tables

Table 1: Problem descriptions

	#ATTR	TRAIN SIZE	TEST SIZE	%POZ
ADULT	106	5000	40222	24%
COV	54	5000	576012	49%
LETTEROPos	16	5000	15000	4%
LETTERAMPos	16	5000	15000	50%

Table 2: Performance over metrics (over four problems)

	ACC	PREC	REC	ROC	F1	MEAN
KNN	.885	.826	.777	.892	.795	.835
SVM	**.899**	**.850**	**.804**	**.936**	**.824**	**.863**
LOGIT	.82	.546	.527	.842	.534	.706

Table 3: Performance by problem (averaged over four metrics)

	COV	ADULT	LetterOPos	LetterAMPos	MEAN
KNN	.777	.696	.914*	.749*	.784
SVM	**.812**	.732	**.937**	**.753**	.807
LOGIT	.769	**.743**	.359	.437	.577

Table 4: Validation performance over single metric (over four problems)

	MEAN ACC
KNN	.882
SVM	.897
LOGIT	.823

Tables Cont.

Table 5: P-Vals for Table 2 (SVM compared with Logit/KNN)

	Pval-Logit	Pval-KNN
SVM-ACC	.021	.0004
SVM-PREC	.019	.0025
SVM-REC	.025	.030
SVM-ROC	.002	.0002
SVM-F1	.023	.0005

Table 6: P-Vals for Table 3(SVM versus rest over data sets)

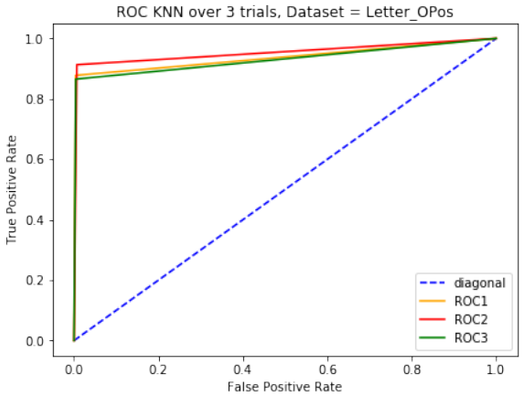
	SVM
COV (KNN)	.045
COV(LOGIT)	.008
ADULT(KNN)	.006
ADULT(LOGIT)	.023
LETTER-O(KNN)	.08
LETTER-O(LOGIT)	5.36e(-5)
LETTER-AM (KNN)	.310
LETTER-AM (LOGIT)	.0001

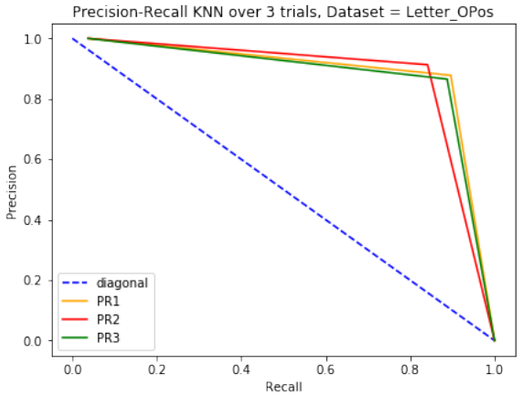
Table 7: Validation scores over data sets (ordered tuple represents score by trial)

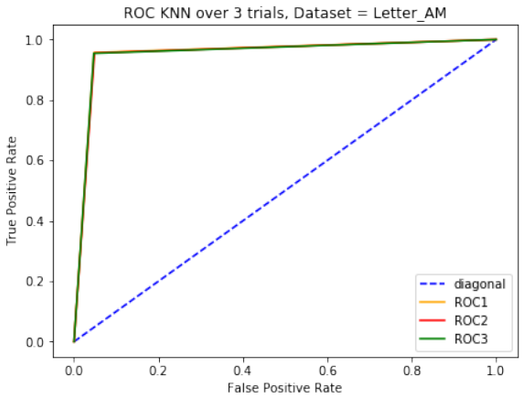
	KNN	SVM	LOGIT
COV	(0.764,0.771 ,0.762)	(0.799,0.793,0.786)	(0.759,0.762,0.753)
ADULT	(0.827,0.835,0.828)	(0.845,0.846,0.844)	(0.841,0.847,0.842)
LETTER-O	(0.988,0.989,0.991)	(0.993,0.992,0.992)	(0.966, 0.962 ,0.961)
LETTER-AM	(0.947,0.943,0.944)	(0.960,0.957,0.954)	(0.736,0.721,0.731)

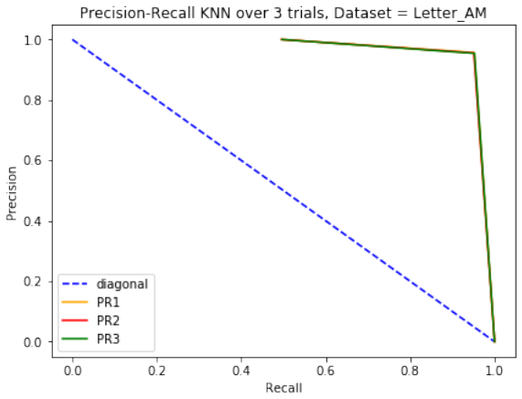
Figures

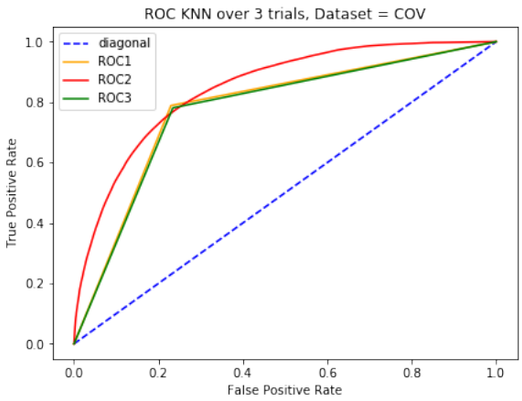
(KNN)

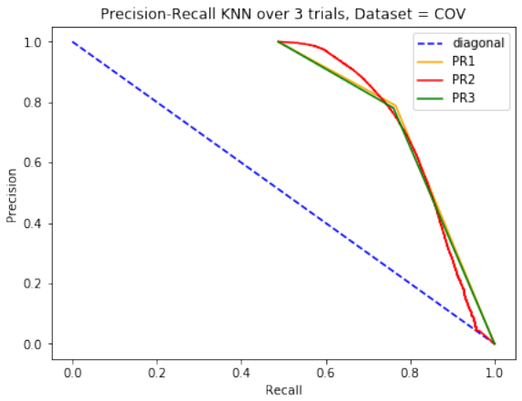

ROC KNN over 3 trials, Dataset = Letter_OPos
True Positive Rate
False Positive Rate
diagonal
ROC1
ROC2
ROC3

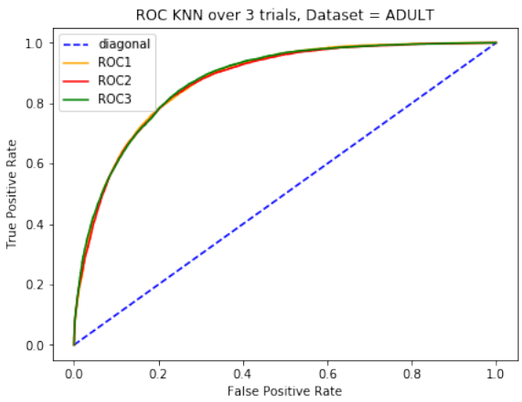

Precision-Recall KNN over 3 trials, Dataset = Letter_OPos
Precision
Recall
diagonal
PR1
PR2
PR3

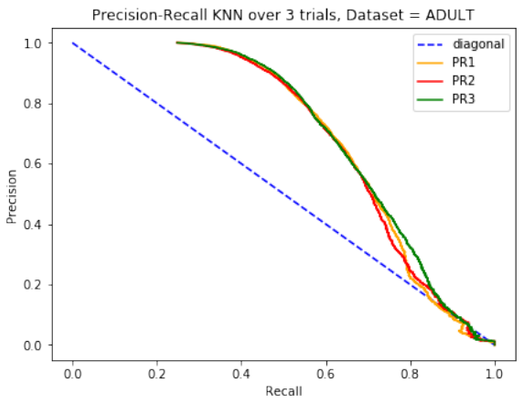

ROC KNN over 3 trials, Dataset = Letter_AM
True Positive Rate
False Positive Rate
diagonal
ROC1
ROC2
ROC3


Precision-Recall KNN over 3 trials, Dataset = Letter_AM
Precision
Recall
diagonal
PR1
PR2
PR3


ROC KNN over 3 trials, Dataset = COV
True Positive Rate
False Positive Rate
diagonal
ROC1
ROC2
ROC3

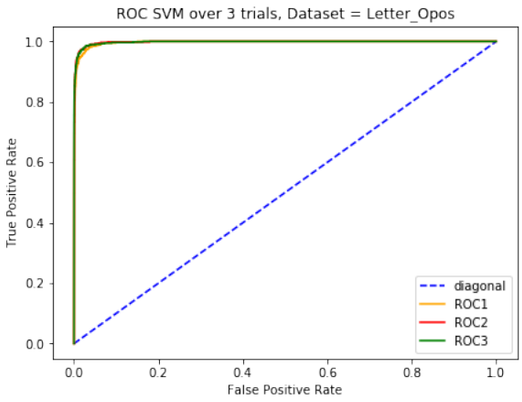

Precision-Recall KNN over 3 trials, Dataset = COV
Precision
Recall
diagonal
PR1
PR2
PR3

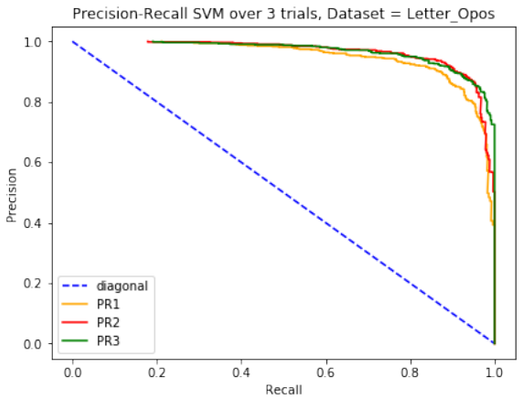

ROC KNN over 3 trials, Dataset = ADULT
True Positive Rate
False Positive Rate
diagonal
ROC1
ROC2
ROC3

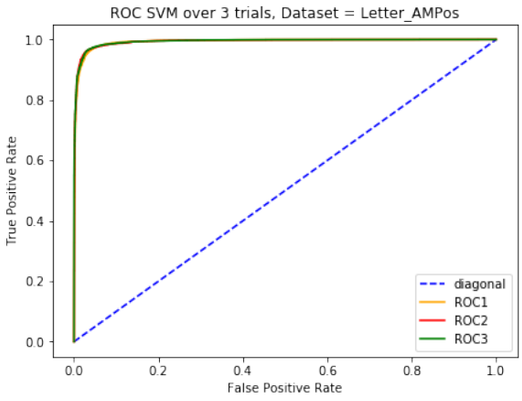

Precision-Recall KNN over 3 trials, Dataset = ADULT
Precision
Recall
diagonal
PR1
PR2
PR3

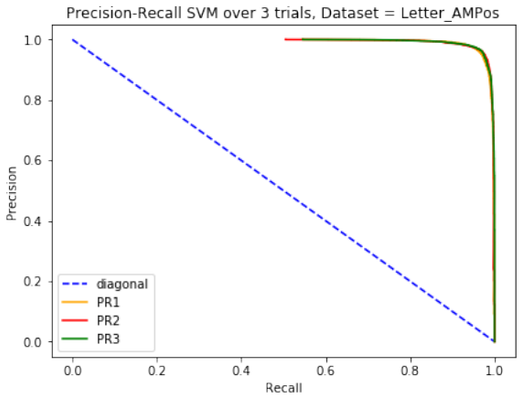
Figures Cont.

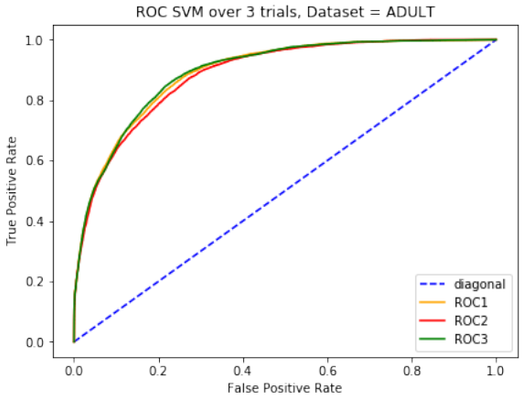
(SVM)

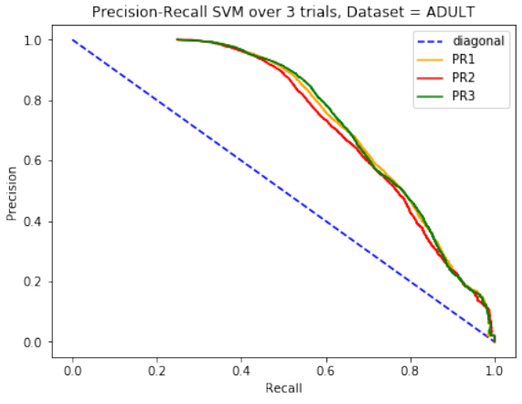


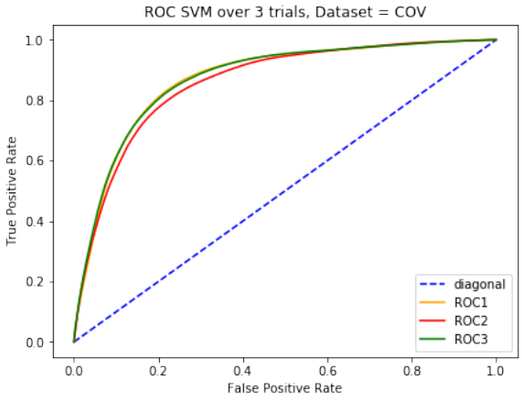


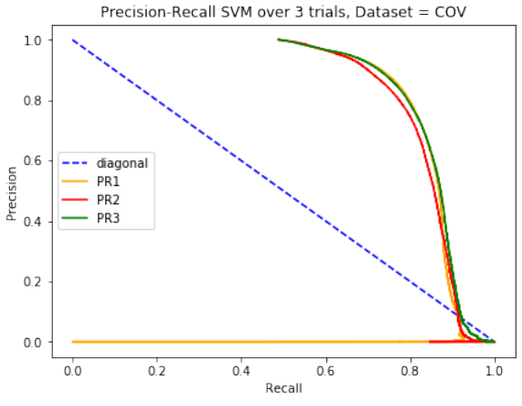






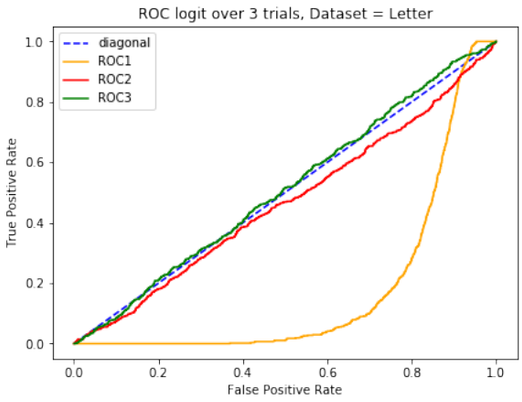


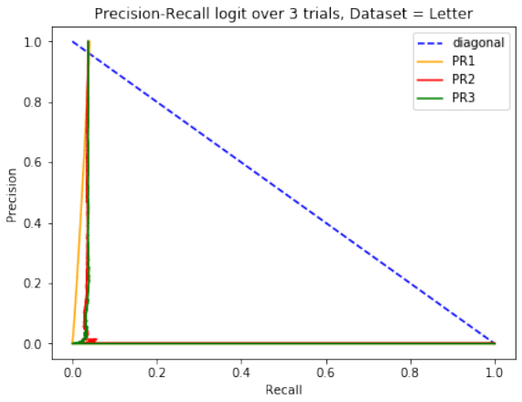


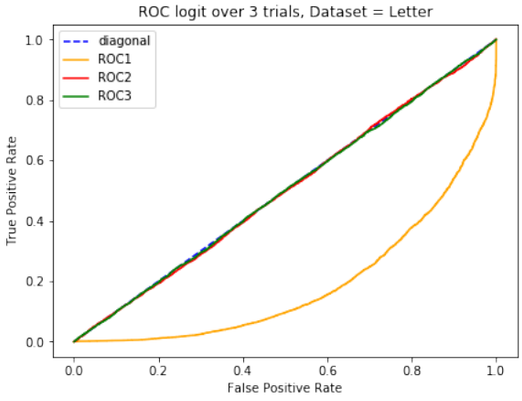


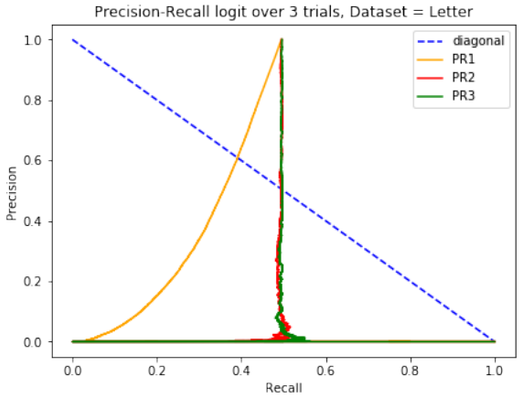
Figures Cont.

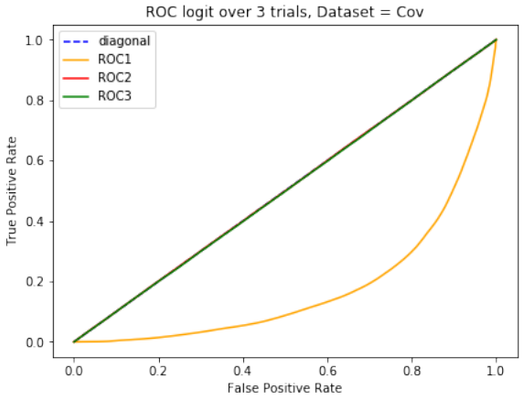
(LOGIT)

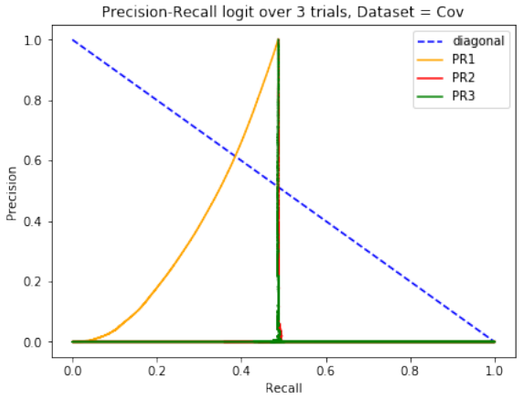


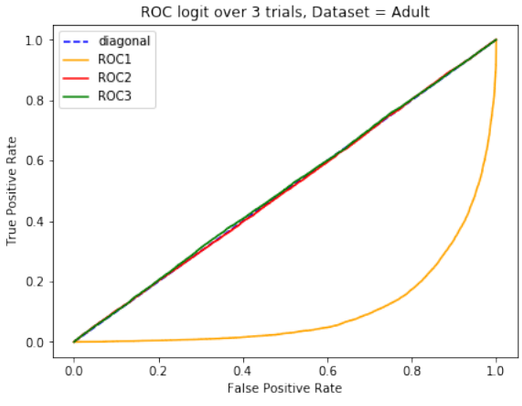


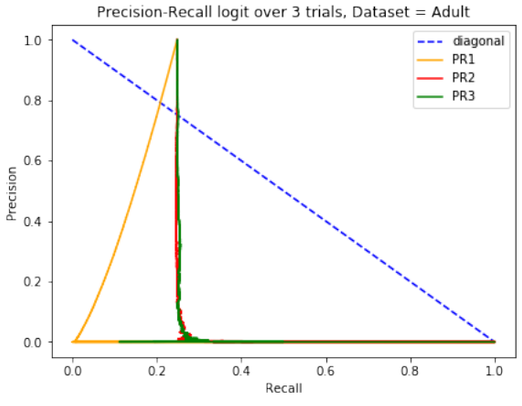












118AProj

December 15, 2020

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import validation_curve
from sklearn.svm import SVC
from sklearn.model_selection import learning_curve
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import log_loss #cross-entropy
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.calibration import CalibratedClassifierCV #platt scaling/isotonic␣
 ↪regression
import pandas as pd
```

[2]:
```
adult_main = pd.read_csv("adult_dat.data")
adult_test = pd.read_csv("adult_test.test")
letters_dat = pd.read_csv("letter-recognition.data")
cov_dat = pd.read_csv("covtype.data")
```

1 Adult Data Processing

[3]:
```
adult = pd.concat([adult_main,adult_test], ignore_index = True)
adult.isnull().values.any()
```

[3]:
```
False
```

[4]:
```
work = pd.get_dummies(adult['workclass']) ##
work.rename(columns = {' ?': 'NA_Work'}, inplace = True)
education = pd.get_dummies(adult['education'])
marital = pd.get_dummies(adult['marital-status'])
occu = pd.get_dummies(adult['occupation']) ##
occu.rename(columns = {' ?': 'NA_Occu'}, inplace = True)
relat = pd.get_dummies(adult['relationship'])
race = pd.get_dummies(adult['race'])
sex = pd.get_dummies(adult['sex'])
country = pd.get_dummies(adult['native-country']) ##
country.rename(columns = {' ?': 'NA_Country'}, inplace = True)
```

[5]:
```
adult_hot= pd.concat([work,education,marital,occu,relat,race,sex,country], axis␣
↪= 1)
```

[6]:
```
adult_hot = pd.
↪concat([adult[['age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week
↪axis =1)
adult['income'] = adult['income'].replace([' >50K.'], ' >50K')
adult['income'] = adult['income'].replace([' <=50K.'], " <=50K")
```

[7]:
```
income_binary = (adult['income'] == ' >50K').astype(int)
adult_hot = pd.concat([adult_hot,income_binary],axis = 1)
adult_hot = adult_hot[adult_hot.NA_Work == 0]
adult_hot = adult_hot[adult_hot.NA_Occu == 0]
adult_hot = adult_hot[adult_hot.NA_Country == 0]
```

[8]:
```
adult_hot.shape
```

[8]:
```
(45222, 109)
```

[9]:
```
adult_hot.drop(['NA_Work','NA_Occu','NA_Country'],axis = 1, inplace=True)
```

[10]:
```
adult_hot.shape
```

[10]:
```
(45222, 106)
```

[11]:
```
adult_hot.reset_index(inplace = True)
```

[88]:
```
len(adult_hot.columns)
```

[88]:
```
107
```

2 Cov Data Processing

[12]:
```
for i in range(1,8,1):
  print(sum((cov_dat['cov']==i).astype(int))) ##check which class is largest = class 2
```

```
211840
283301
35754
2747
9493
17367
20510
```

[13]:
```
class2 = pd.DataFrame((cov_dat['cov'] == 2).astype(int))
class2.rename(columns = {'cov':'class2'}, inplace = True)
```

[14]:
```
del cov_dat['cov']
cov_bin = pd.concat([cov_dat,class2], axis = 1)
```

[15]:
```
cov_bin.isnull().any(); #check nullity
```

3 Letter Data Processing

[17]:
```
switch = letters_dat['lettr']
del letters_dat['lettr']
letters_dat = pd.concat([letters_dat,switch],1)
```

[18]:
```
O_pos = pd.DataFrame((letters_dat['lettr'] == 'O').astype(int))
O_pos = O_pos.rename(columns = {'lettr':'O_pos'})
```

[19]:
```
letters_dat = pd.concat([letters_dat,O_pos],axis = 1)
```

[20]:
```
AM_pos = pd.DataFrame((letters_dat['lettr'] == 'A').astype(int))
for i in ['B','C','D','E','F','G','H','I','J','K','L','M']:
      AM_pos = AM_pos + pd.DataFrame((letters_dat['lettr'] == i).astype(int))
```

[21]:
```
AM_pos = AM_pos.rename(columns = {'lettr':'AM_pos'})
```

[22]:
```
letters_dat = pd.concat([letters_dat,AM_pos], axis = 1)
```

[23]:
```
del letters_dat['Unnamed: 17']
```

[24]:
```
letters_dat.shape
```

[24]: (20000, 19)

3.0.1 Function for plotting ROC/Precision-Recall with KNN/Logit

[25]:
```
##different than SVM since we have to use best_fit.decision_function(..)
##predict_proba with SVM is both computationally expensive and doesn't give␣
 ↪consistent results
```

[26]:
```
def roc_PR_plot(results, dataset_string, classifier_string):
    fig, ax = plt.subplots(nrows = 1, ncols = 2, figsize = (15,5))
    tru_ys = results.get('y_true')
    probabilities_pos = results.get('tst_prob').loc[:,1]
    fp1, tp1, thresholds1 = roc_curve(tru_ys.get('y1'), probabilities_pos.iloc[:
 ↪,0])
    fp2, tp2 , thresholds2 = roc_curve(tru_ys.get('y2'), probabilities_pos.
 ↪iloc[:,1])
    fp3, tp3 , thresholds3 = roc_curve(tru_ys.get('y3'), probabilities_pos.
 ↪iloc[:,2])
    ax[0].plot([0, 1], [0, 1], color='blue', linestyle='--', label = "diagonal")
    ax[0].plot(fp1, tp1, color='orange', label='ROC1')
    ax[0].plot(fp2, tp2, color='red', label='ROC2')
    ax[0].plot(fp3, tp3, color='green', label='ROC3')
    ax[0].set_xlabel('False Positive Rate')
    ax[0].set_ylabel('True Positive Rate')
    ax[0].set_title('ROC ' + classifier_string + ' over 3 trials, Dataset = ' +␣
 ↪dataset_string)
    ax[0].legend()
    f1, t1, thres1 = precision_recall_curve(tru_ys.get('y1'), probabilities_pos.
 ↪iloc[:,0])
    f2, t2 , thresh2 = precision_recall_curve(tru_ys.get('y2'),␣
 ↪probabilities_pos.iloc[:,1])
    f3, t3 , thresholds3 = precision_recall_curve(tru_ys.get('y3'),␣
 ↪probabilities_pos.iloc[:,2])
```

```
    ax[1].plot([0, 1], [1,0], color='blue', linestyle='--', label = "diagonal")
    ax[1].plot(f1, t1, color='orange', label='PR1')
    ax[1].plot(f2, t2, color='red', label='PR2')
    ax[1].plot(f3, t3, color='green', label='PR3')
    ax[1].set_xlabel('Recall')
    ax[1].set_ylabel('Precision')
    ax[1].set_title('Precision-Recall ' + classifier_string + ' over 3 trials,␣
↪Dataset = ' + dataset_string)
    ax[1].legend()
    plt.show()
```

4 Begin KNN Trials

[75]:
```
def KNN(dataset, target,lower,upper,test_size, scaler):
    grid = np.arange(1,500,20)
    tst_predictions = pd.DataFrame()
    validation_scores = np.zeros(3)
    tst_prob = pd.DataFrame()
    params_best = {}
    y_true = {}
    acc_scores = np.zeros(3)
    precision_scores = np.zeros(3)
    recall_scores = np.zeros(3)
    roc_scores_1 = np.zeros(3)
    f1_scores = np.zeros(3)


    for j in [1,2,3]:

        X_tr, X_tst, y_tr, y_tst = train_test_split(dataset.iloc[:,lower:
↪upper], dataset.loc[:,target], test_size= test_size, random_state = j,␣
↪shuffle = True)
        sca = scaler
        X_tr = sca.fit_transform(X_tr)
        X_tst = sca.transform(X_tst)
        five_fold = KFold(n_splits = 5, shuffle = True)
        knn = KNeighborsClassifier()
        parameters = {'n_neighbors': list(grid), 'weights':␣
↪['uniform','distance']}

        print(X_tr.shape[0]) #print statement for progress
        GS = GridSearchCV(estimator = knn, param_grid = parameters, cv =␣
↪five_fold, n_jobs = 4, verbose = 0, refit = True)
        best_fit = GS.fit(X_tr,y_tr)
        validation_scores[j-1]= best_fit.best_score_
```

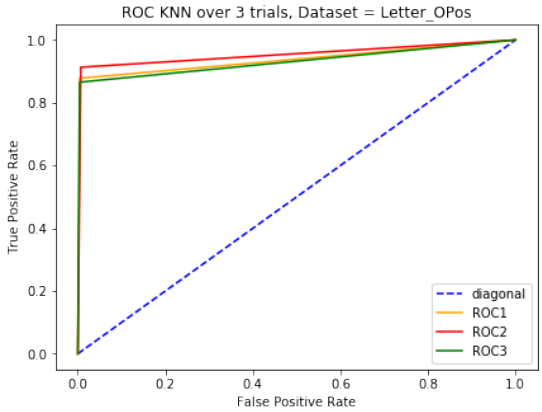
```
        print(best_fit.best_score_) #print statement for progress
        predictions = best_fit.predict(X_tst)
        prediction_prob = best_fit.predict_proba(X_tst)

        tst_predictions = pd.concat([tst_predictions, pd.DataFrame(predictions,␣
↪columns = [str(j)])], axis = 1)
        tst_prob = pd.concat([tst_prob,pd.DataFrame(prediction_prob).
↪reset_index(drop = True)], axis = 1)
        params_best['best' + str(j)]= best_fit.best_params_
        y_true['y' + str(j)] = y_tst
        acc_scores[j-1] = accuracy_score(y_tst, predictions)
        precision_scores[j-1] = precision_score(y_tst, predictions)
        recall_scores[j-1] = recall_score(y_tst, predictions)
        roc_scores_1[j-1] = roc_auc_score(y_tst, prediction_prob[:,1])
        f1_scores[j-1] = f1_score(y_tst,predictions)

    dict_of_vals = {'best_validation': validation_scores, 'best_params':␣
↪params_best, 'tst_predictions': tst_predictions, 'tst_prob': tst_prob,␣
↪'y_true': y_true, 'acc_scores': acc_scores, 'precision_scores':␣
↪precision_scores, 'recall_scores': recall_scores, 'roc_scores':␣
↪roc_scores_1, 'f1_scores': f1_scores}




    return dict_of_vals
```

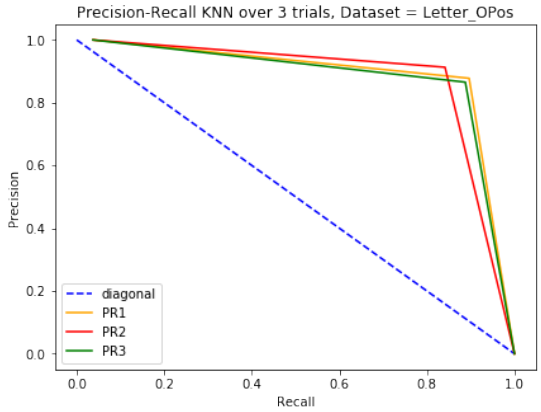
4.1 Letter Dat

[76]:
```
results = KNN(letters_dat,'O_pos',0,16,.75, StandardScaler()) #unbalanced
```

```
5000
0.9884
5000
0.9888
5000
0.9906
```

[29]:
```
roc_PR_plot(results, 'Letter_OPos', 'KNN')
```

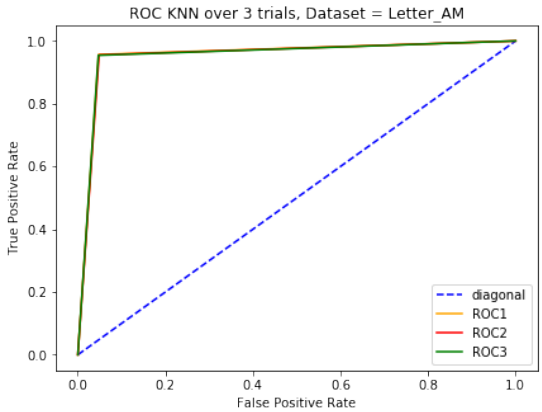


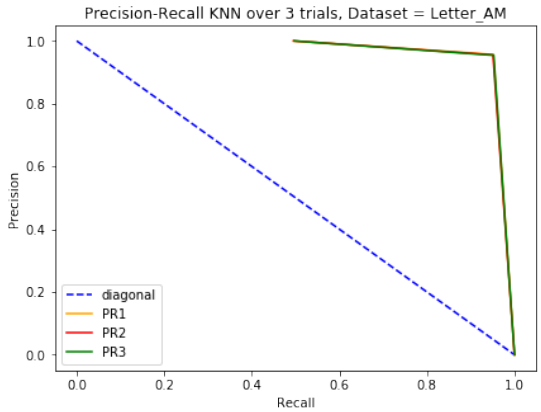


[30]:
```
results_AM = KNN(letters_dat,'AM_pos',0,16,.75, StandardScaler()) #balanced
```

```
5000
0.9468
5000
0.943
5000
0.944
```

[32]:
```
roc_PR_plot(results_AM, 'Letter_AM', 'KNN')
```





4.2 Cov Dat

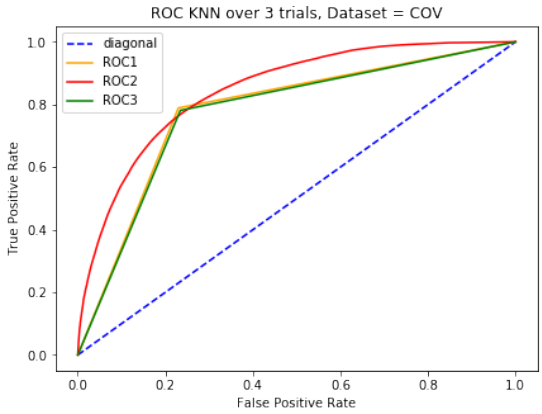
[28]:
```
results_cov = KNN(cov_bin, 'class2', 0,54, 576012, StandardScaler())
```

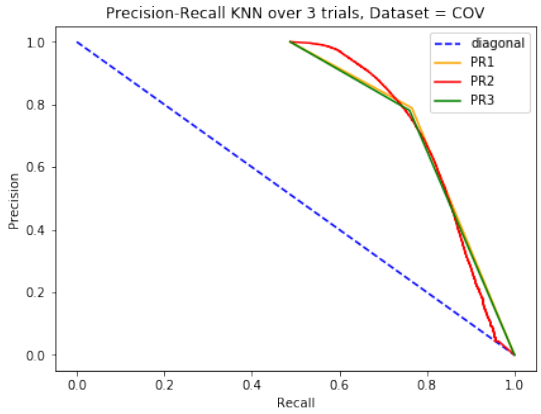
```
5000
0.7644
```

```
5000
0.771
5000
0.7616
```

[30]:
```
roc_PR_plot(results_cov, 'COV', 'KNN')
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/pylabtools.py:128:
UserWarning: Creating legend with loc="best" can be slow with large amounts of
data.
  fig.canvas.print_figure(bytes_io, **kw)
```



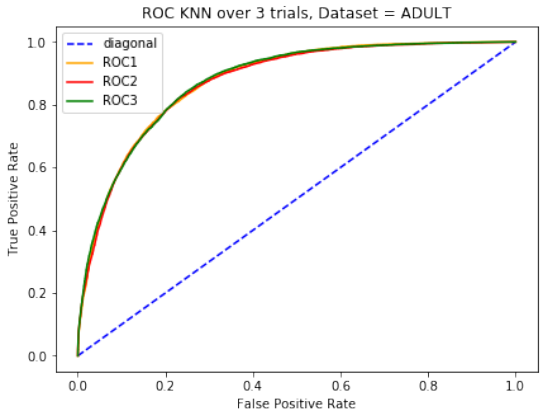


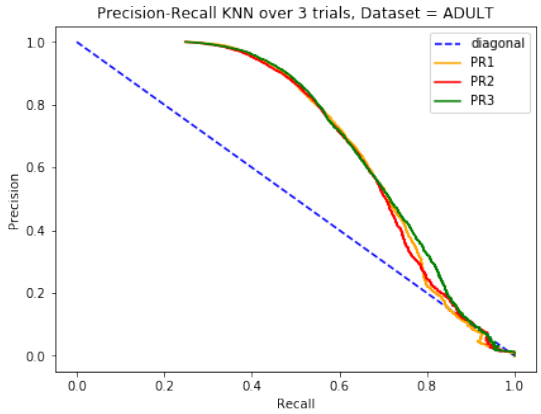
4.3 Adult Dat

[31]:
```
results_adult = KNN(adult_hot, 'income', 0, 105, 40222, StandardScaler())
```

```
5000
0.8272
5000
0.8346
5000
0.8278
```

[32]:
```
roc_PR_plot(results_adult, 'ADULT', 'KNN')
```





5 Begin SVM Trials

[78]:
```
def roc_PR_plot_SVM(results, dataset_string, classifier_string):
    fig, ax = plt.subplots(nrows = 1, ncols = 2, figsize = (15,5))
    tru_ys = results.get('y_true')
    probabilities_pos = results.get('tst_prob')
    fp1, tp1, thresholds1 = roc_curve(tru_ys.get('y1'), probabilities_pos.iloc[:
↪,0])
    fp2, tp2 , thresholds2 = roc_curve(tru_ys.get('y2'), probabilities_pos.
↪iloc[:,1])
    fp3, tp3 , thresholds3 = roc_curve(tru_ys.get('y3'), probabilities_pos.
↪iloc[:,2])
    ax[0].plot([0, 1], [0, 1], color='blue', linestyle='--', label = "diagonal")
    ax[0].plot(fp1, tp1, color='orange', label='ROC1')
    ax[0].plot(fp2, tp2, color='red', label='ROC2')
    ax[0].plot(fp3, tp3, color='green', label='ROC3')
    ax[0].set_xlabel('False Positive Rate')
    ax[0].set_ylabel('True Positive Rate')
    ax[0].set_title('ROC ' + classifier_string + ' over 3 trials, Dataset = ' +␣
↪dataset_string)
    ax[0].legend()
    f1, t1, thres1 = precision_recall_curve(tru_ys.get('y1'), probabilities_pos.
↪iloc[:,0])
    f2, t2, thresh2 = precision_recall_curve(tru_ys.get('y2'),␣
↪probabilities_pos.iloc[:,1])
    f3, t3 , thres3 = precision_recall_curve(tru_ys.get('y3'),␣
↪probabilities_pos.iloc[:,2])
    ax[1].plot([0, 1], [1,0], color='blue', linestyle='--', label = "diagonal")
    ax[1].plot(f1, t1, color='orange', label='PR1')
    ax[1].plot(f2, t2, color='red', label='PR2')
```

```
    ax[1].plot(f3, t3, color='green', label='PR3')
    ax[1].set_xlabel('Recall')
    ax[1].set_ylabel('Precision')
    ax[1].set_title('Precision-Recall ' + classifier_string + ' over 3 trials,␣
↪Dataset = ' + dataset_string)
    ax[1].legend()
    plt.show()
```

[58]:
```
def SVM(dataset, target,lower,upper,test_size, scaler):

    import re

    tst_predictions = pd.DataFrame()
    validation_scores = np.zeros(3)
    tst_prob = pd.DataFrame()
    params_best = {}
    y_true = {}
    acc_scores = np.zeros(3)
    precision_scores = np.zeros(3)
    recall_scores = np.zeros(3)
    roc_scores_1 = np.zeros(3)
    f1_scores = np.zeros(3)


    def unpack_more(params):
        new_dict = {}
        test_dict = params
        for key in test_dict.keys() :
            new_dict[re.sub('classifier__', '', key)] = (test_dict[key])
        return new_dict



    for j in [1,2,3]:

        pipe = Pipeline([('std', scaler), ('classifier', SVC())])
        param_grid = [{'classifier__kernel': ['rbf'], 'classifier__C': np.
↪power(10., np.arange(-7, 4)),
                       'classifier__gamma':[0.001,0.005,0.01,0.05,0.1,0.5,1,2]},
                  {'classifier__kernel': ['poly'], 'classifier__C': np.power(10.
↪, np.arange(-7, 4)), 'classifier__degree': [2,3]},
                    {'classifier__kernel': ['linear'], 'classifier__C': np.
↪power(10., np.arange(-7, 4))}]
```

```
        X_tr, X_tst, y_tr, y_tst = train_test_split(dataset.iloc[:,lower:
↪upper], dataset.loc[:,target], test_size= test_size, random_state = j,␣
↪shuffle = True)
        five_fold = KFold(n_splits = 5, shuffle = True)

        print(X_tr.shape[0]) ##print statement to track progress
        GS = GridSearchCV(estimator = pipe, param_grid = param_grid, cv =␣
↪five_fold, n_jobs = -1, verbose = 0, refit = True)
        best_fit = GS.fit(X_tr,y_tr)
        print(best_fit.best_score_) ##print statement to track progress
        validation_scores[j-1]= best_fit.best_score_
        predictions = best_fit.predict(X_tst)
        decision = best_fit.decision_function(X_tst)
        prediction_prob = np.exp(decision)/(np.exp(decision)+1) ##bad practice

        tst_predictions = pd.concat([tst_predictions, pd.DataFrame(predictions,␣
↪columns = [str(j)])], axis = 1)
        tst_prob = pd.concat([tst_prob,pd.DataFrame(prediction_prob).
↪reset_index(drop = True)], axis = 1)
        params_best['best' + str(j)]= best_fit.best_params_
        y_true['y' + str(j)] = y_tst
        acc_scores[j-1] = accuracy_score(y_tst, predictions)
        precision_scores[j-1] = precision_score(y_tst, predictions)
        recall_scores[j-1] = recall_score(y_tst, predictions)
        roc_scores_1[j-1] = roc_auc_score(y_tst, prediction_prob)
        f1_scores[j-1] = f1_score(y_tst,predictions)

    dict_of_vals = {'best_validation': validation_scores, 'best_params':␣
↪params_best, 'tst_predictions': tst_predictions, 'tst_prob': tst_prob,␣
↪'y_true': y_true, 'acc_scores': acc_scores, 'precision_scores':␣
↪precision_scores, 'recall_scores': recall_scores, 'roc_scores':␣
↪roc_scores_1, 'f1_scores': f1_scores}




    return dict_of_vals
```

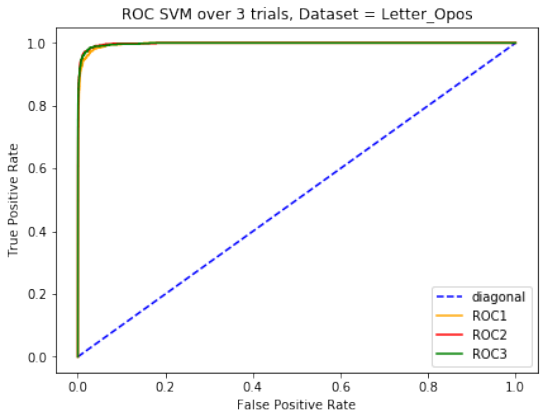
5.1 Letter Data

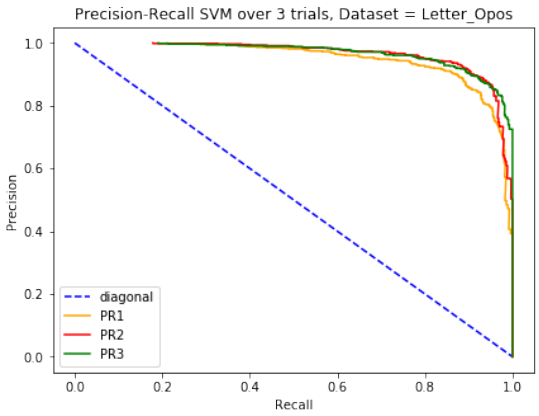
[59]:
```
letter_SVM = SVM(letters_dat, 'O_pos', 0,16,.75, StandardScaler())
```

```
5000
0.9934
5000
0.9924
```

```
5000
0.9924
```

[81]:
```
roc_PR_plot_SVM(letter_SVM, 'Letter_Opos', 'SVM')
```



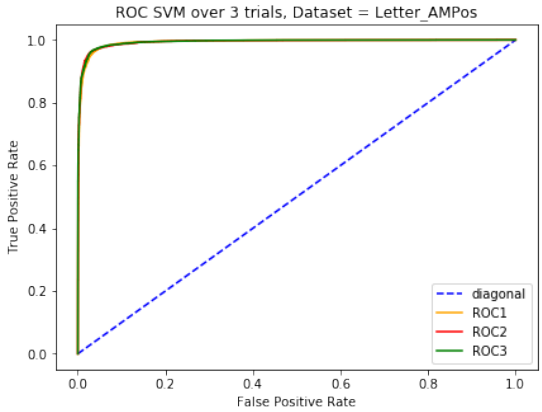


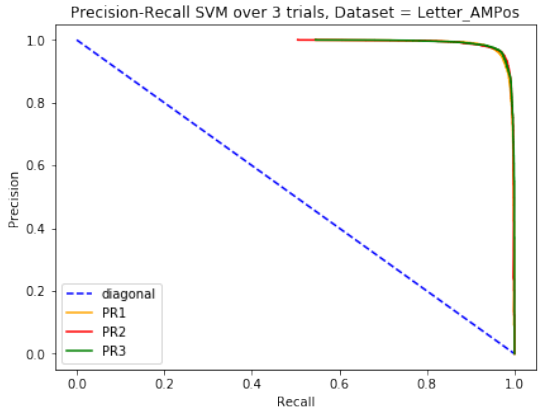
[62]:
```
letter_SVM_AM = SVM(letters_dat, 'AM_pos', 0,16,.75, StandardScaler())
```

```
5000
0.9602
5000
0.9572
5000
0.9542
```

[69]:
```
AM_svm_val = np.array([0.9602, 0.9572, 0.9542])
AM_svm_acc = np.array([0.9624    , 0.96606667, 0.96606667])
AM_svm_prec = np.array([0.96103198, 0.96797297, 0.96727273])
AM_svm_recall = np.array([0.96310261, 0.96341627, 0.96428571])
AM_svm_roc = np.array([0.99276918, 0.99314926, 0.993619  ])
AM_svm_f1 = np.array([0.96206618, 0.96568925, 0.96577691])
```

[82]:
```
roc_PR_plot_SVM(letter_SVM_AM, 'Letter_AMPos', 'SVM')
```



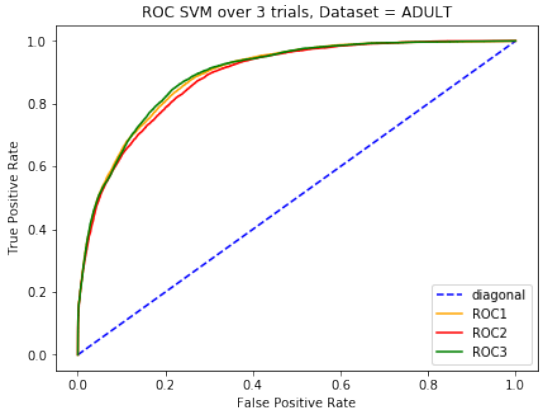


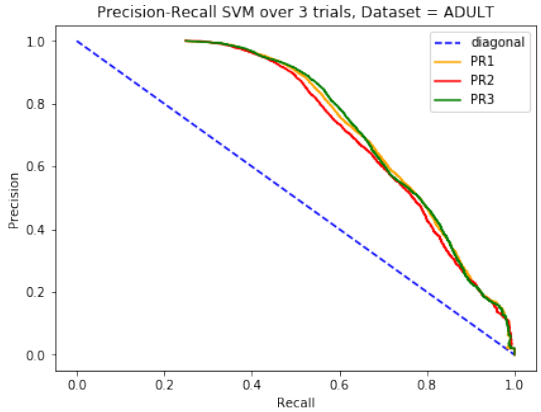
5.2 Adult Data

[63]:
```
adult_svm = SVM(adult_hot, 'income', 0, 105, 40222, StandardScaler())
```

```
5000
0.8452
5000
0.846
5000
0.8444
```

[83]:
```
roc_PR_plot_SVM(adult_svm, 'ADULT', 'SVM')
```





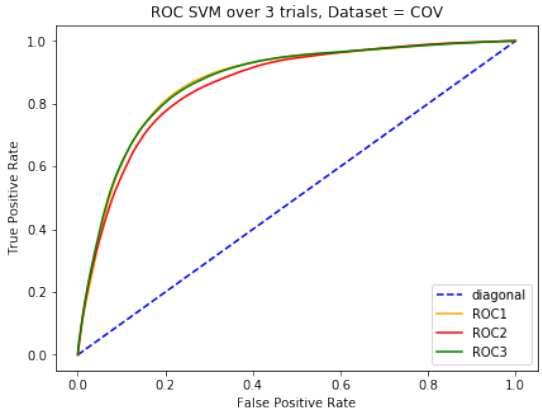
5.3 Cov Data

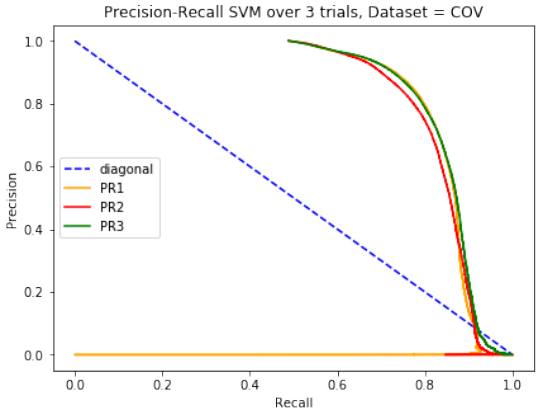
[65]:
```
cov_svm = SVM(cov_bin, 'class2', 0,54, 576012, StandardScaler())
```

```
5000
0.799
5000
0.7926
5000
0.7862
```

[84]:
```
roc_PR_plot_SVM(cov_svm, 'COV', 'SVM')
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/pylabtools.py:128:
UserWarning: Creating legend with loc="best" can be slow with large amounts of
data.
  fig.canvas.print_figure(bytes_io, **kw)
```





6 Begin Logit Trials

[69]:
```
def Logistic(dataset, target,lower,upper,test_size,scaler):

    tst_predictions = pd.DataFrame()
    validation_scores = np.zeros(3)
    tst_prob = pd.DataFrame()
    params_best = {}
    y_true = {}
    acc_scores = np.zeros(3)
    precision_scores = np.zeros(3)
    recall_scores = np.zeros(3)
    roc_scores_1 = np.zeros(3)
```

```
   f1_scores = np.zeros(3)


   for j in [1,2,3]:

       pipe = Pipeline([('std', scaler), ('classifier',␣
↪LogisticRegression(solver = 'lbfgs', max_iter = 1000))])
       param_grid = [{'classifier__penalty': ['l2'], 'classifier__C': np.
↪power(10., np.arange(-8, 5))},
                {'classifier__penalty': ['none'],'classifier__C': np.power(10.,␣
↪np.arange(-8, 5))}]



       X_tr, X_tst, y_tr, y_tst = train_test_split(dataset.iloc[:,lower:
↪upper], dataset.loc[:,target], test_size= test_size, random_state = j,␣
↪shuffle = True)
       five_fold = KFold(n_splits = 5, shuffle = True)

       print(X_tr.shape[0]) #print statement to check progress
       GS = GridSearchCV(estimator = pipe, param_grid = param_grid, cv =␣
↪five_fold, n_jobs = 4, verbose = 0, refit = True)
       best_fit = GS.fit(X_tr,y_tr)
       validation_scores[j-1]= best_fit.best_score_
       print(best_fit.best_score_) #print statement to check progress
       predictions = best_fit.predict(X_tst)
       prediction_prob = best_fit.predict_proba(X_tst)


       tst_predictions = pd.concat([tst_predictions, pd.DataFrame(predictions,␣
↪columns = [str(j)])], axis = 1)
       tst_prob = pd.concat([tst_prob,pd.DataFrame(prediction_prob).
↪reset_index(drop = True)], axis = 1)
       params_best['best' + str(j)]= best_fit.best_params_
       y_true['y' + str(j)] = y_tst
       acc_scores[j-1] = accuracy_score(y_tst, predictions)
       precision_scores[j-1] = precision_score(y_tst, predictions)
       recall_scores[j-1] = recall_score(y_tst, predictions)
       roc_scores_1[j-1] = roc_auc_score(y_tst, prediction_prob[:,1])
       f1_scores[j-1] = f1_score(y_tst,predictions)

   dict_of_vals = {'best_validation': validation_scores, 'best_params':␣
↪params_best, 'tst_predictions': tst_predictions, 'tst_prob': tst_prob,␣
↪'y_true': y_true, 'acc_scores': acc_scores, 'precision_scores':␣
↪precision_scores, 'recall_scores': recall_scores, 'roc_scores':␣
↪roc_scores_1, 'f1_scores': f1_scores}
```

```
    return dict_of_vals
```

[70]:
```
results_O_log = Logistic(letters_dat,'O_pos',0,16,.75, StandardScaler())␣
↪#unbalanced
```

```
5000
0.9656
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/metrics/classification.py:1437:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples.
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.7/site-packages/sklearn/metrics/classification.py:1437:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

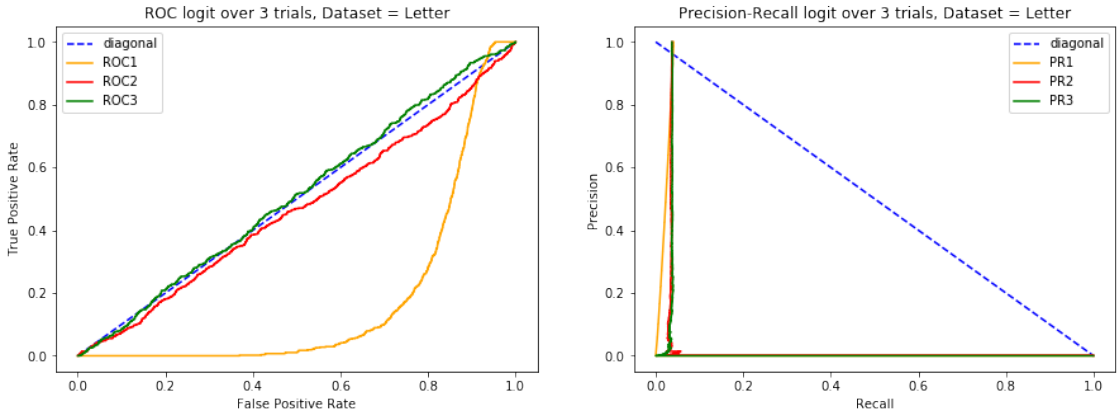
```
5000
0.962
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/metrics/classification.py:1437:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples.
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.7/site-packages/sklearn/metrics/classification.py:1437:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
5000
0.9608
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/metrics/classification.py:1437:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples.
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.7/site-packages/sklearn/metrics/classification.py:1437:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

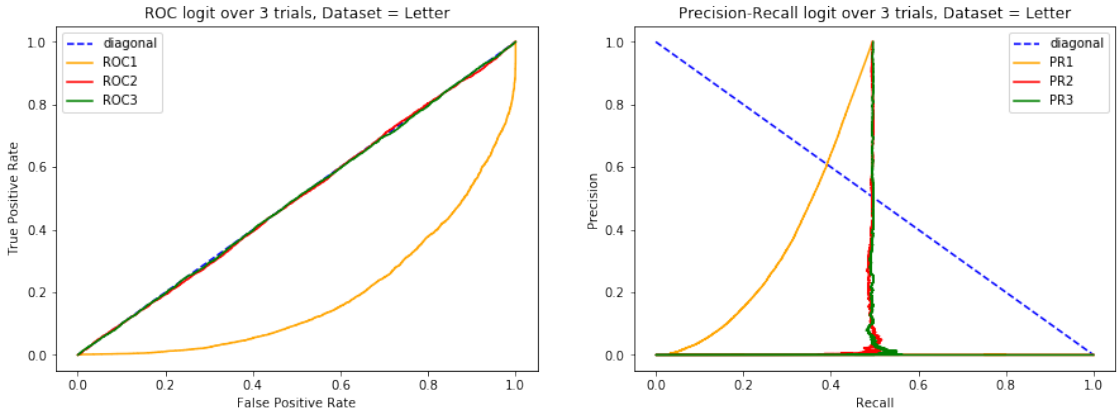
[72]:
```
roc_PR_plot_SVM(results_O_log, 'Letter', 'logit')
```



[73]:
```
results_AM_log = Logistic(letters_dat,'AM_pos',0,16,.75, StandardScaler())␣
↪#balanced
```

```
5000
0.736
5000
0.7206
5000
0.731
```

[75]:
```
roc_PR_plot_SVM(results_AM_log, 'Letter', 'logit')
```



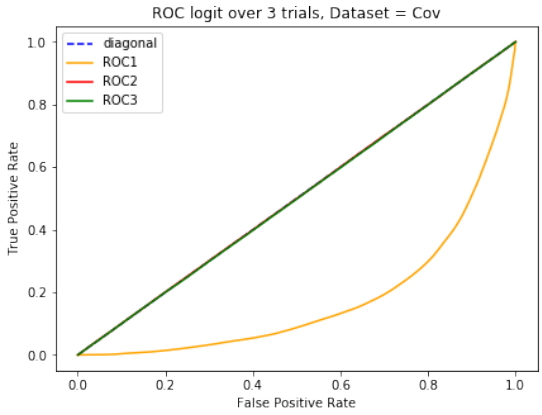
[76]:
```
cov_log= Logistic(cov_bin, 'class2', 0,54, 576012, StandardScaler())
```

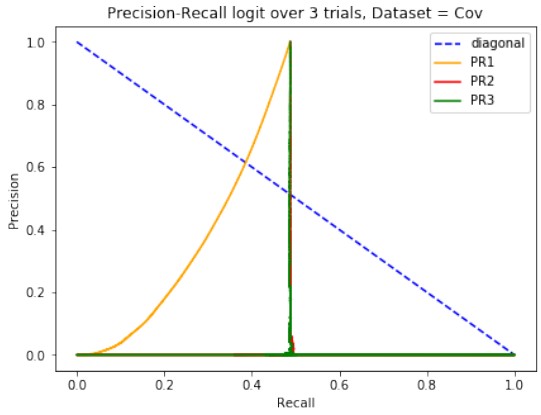
```
5000
0.7592
5000
0.7616
```

```
5000
0.7532
```

[78]:
```
roc_PR_plot_SVM(cov_log, 'Cov', 'logit')
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/pylabtools.py:128:
UserWarning: Creating legend with loc="best" can be slow with large amounts of
data.
  fig.canvas.print_figure(bytes_io, **kw)
```

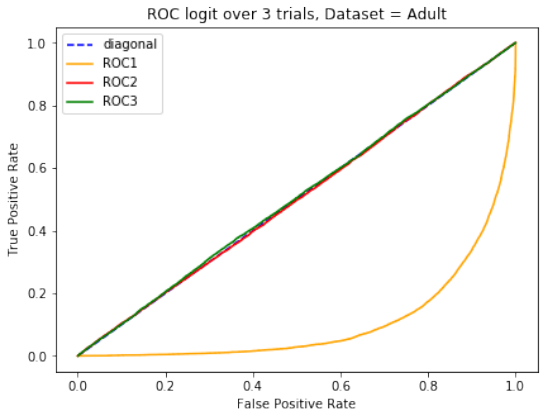


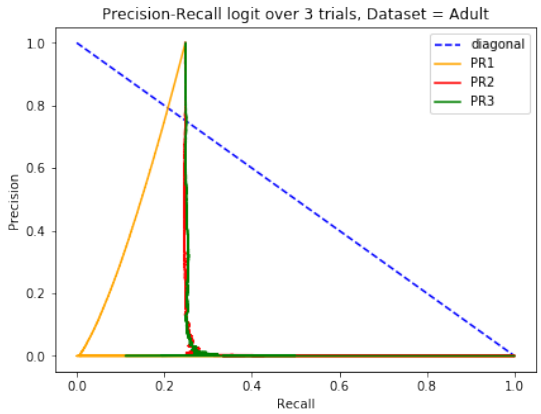


[79]:
```
adult_log = Logistic(adult_hot, 'income', 0, 105, 40222, StandardScaler())
```

```
5000
0.841
5000
0.8466
5000
0.8418
```

[81]:
```
roc_PR_plot_SVM(adult_log, 'Adult', 'logit')
```





6.1 ALL RESULTS.

6.1.1 –had to retrieve them manully from the returned dictionary since jupyter lab shut down

[2]:
```
##Need O_pos KNN
## Need all results from SVM

O_knn_val = np.array([0.9884, 0.9888, 0.9906])
O_knn_acc = np.array([0.99133333, 0.99026667, 0.99093333])
O_knn_prec = np.array([0.89630931, 0.84124386, 0.88766114])
O_knn_recall = np.array([0.8777969 , 0.91296625, 0.86535009])
O_knn_roc = np.array([0.93685254, 0.9531237 , 0.9305633 ])
O_knn_f1 = np.array([0.88695652, 0.87563884, 0.87636364])

cov_knn_val = np.array([0.7644, 0.771 , 0.7616])
cov_knn_acc = np.array([0.77929279, 0.76678784, 0.77316792])
cov_knn_prec = np.array([0.76566599, 0.75334624, 0.76028943])
cov_knn_recall = np.array([0.78872482, 0.77579572, 0.7809207 ])
cov_knn_roc = np.array([0.77952156, 0.84640168, 0.77335698])
cov_knn_f1 = np.array([0.77702437, 0.76440619, 0.77046698])

adult_knn_val = np.array([0.8272, 0.8346, 0.8278])
adult_knn_acc = np.array([0.82442444, 0.82474765, 0.82526975])
adult_knn_prec = np.array([0.71873596, 0.70062112, 0.73140367])
adult_knn_recall = np.array([0.48071722, 0.50984331, 0.46990602])
adult_knn_roc = np.array([0.8749578 , 0.87187256, 0.87591739])
adult_knn_f1 = np.array([0.57611044, 0.59019824, 0.57219382])

AM_knn_val = np.array([0.9468, 0.943 , 0.944 ])
AM_knn_acc = np.array([0.9546    , 0.95346667, 0.9536])
AM_knn_prec = np.array([0.95177495, 0.95087672, 0.95254692])
AM_knn_recall = np.array([0.9567735 , 0.95548083, 0.95408163])
AM_knn_roc = np.array([0.95462124, 0.95348397, 0.95360332])
AM_knn_f1 = np.array([0.95426768, 0.95317322, 0.95331366])

cov_log_val = np.array([0.7592, 0.7616, 0.7532])
cov_log_acc = np.array([0.75468567, 0.75341312, 0.75970987])
cov_log_prec = np.array([0.73840708, 0.74472044, 0.74148755])
cov_log_recall = np.array([0.7694581 , 0.75221513, 0.77852404])
cov_log_roc = np.array([0.82486558, 0.82433487, 0.82397922])
cov_log_f1 = np.array([0.75361288, 0.74844903, 0.75955458])
```

```
AM_log_val = np.array([0.736 , 0.7206, 0.731 ])
AM_log_acc = np.array([0.7262, 0.7258, 0.727 ])
AM_log_prec = np.array([0.71240241, 0.7159948 , 0.71685422])
AM_log_recall = np.array([0.74952868, 0.74055145, 0.74409237])
AM_log_roc = np.array([0.81066856, 0.8131976 , 0.8124264 ])
AM_log_f1 =  np.array([0.73049413, 0.72806612, 0.73021938])

adult_log_val = np.array([0.841 , 0.8466, 0.8418])
adult_log_acc = np.array([0.84456268, 0.84391626, 0.84289692])
adult_log_prec = np.array([0.72853118, 0.7222088 , 0.72681365])
adult_log_recall = np.array([0.59571271, 0.6003415 , 0.589982  ])
adult_log_roc = np.array([0.89477575, 0.89843187, 0.89665055])
adult_log_f1 = np.array([0.65546126, 0.65566038, 0.65128856])

O_log_val = np.array([0.9656, 0.962 , 0.9608])
O_log_acc = np.array([0.96126667, 0.96246667, 0.96286667])
O_log_prec = np.array([0,0,0])
O_log_recall = np.array([0,0,0])
O_log_roc = np.array([0.82614902, 0.83762008, 0.84356906])
O_log_f1 = np.array([0., 0., 0.])

cov_svm_val = np.array([0.799 , 0.7926, 0.7862])
cov_svm_acc = np.array([0.80562558, 0.78864329, 0.8025805 ])
cov_svm_prec = np.array([0.7852103 , 0.77481568, 0.78118153])
cov_svm_recall = np.array([0.82777455, 0.79874621, 0.82657127])
cov_svm_roc = np.array([0.8693234 , 0.85813217, 0.87016745])
cov_svm_f1 = np.array([0.80593083, 0.78659898, 0.80323569])

adult_svm_val = np.array([0.8452, 0.846 , 0.8444])
adult_svm_acc = np.array([0.84222565, 0.83822286, 0.83926707])
adult_svm_prec = np.array([0.73786789, 0.7174379 , 0.73489175])
adult_svm_recall = np.array([0.5650606 , 0.57151466, 0.55318936])
adult_svm_roc = np.array([0.89260571, 0.88584867, 0.89383485])
adult_svm_f1 = np.array([0.64000454, 0.63621625, 0.63122469])

O_svm_val = np.array([0.9934, 0.9924, 0.9924])
O_svm_acc = np.array([0.99113333, 0.9932    , 0.99326667])
O_svm_prec = np.array([0.91481481, 0.92449355, 0.94357977])
O_svm_recall = np.array([0.85025818, 0.89165187, 0.87073609])
O_svm_roc = np.array([0.99675533, 0.99784277, 0.99772609])
O_svm_f1 = np.array([0.88135593, 0.90777577, 0.90569561])

AM_svm_val = np.array([0.9602, 0.9572, 0.9542])
AM_svm_acc = np.array([0.9624    , 0.96606667, 0.96606667])
AM_svm_prec = np.array([0.96103198, 0.96797297, 0.96727273])
AM_svm_recall = np.array([0.96310261, 0.96341627, 0.96428571])
```

```
AM_svm_roc = np.array([0.99276918, 0.99314926, 0.993619  ])
AM_svm_f1 = np.array([0.96206618, 0.96568925, 0.96577691])
```

[3]:
```
# table 4

KNN_val = (sum(O_knn_val) + sum(cov_knn_val) + sum(adult_knn_val) +␣
 ↪sum(AM_knn_val))/12
SVM_val = (sum(O_svm_val) + sum(cov_svm_val) + sum(adult_svm_val) +␣
 ↪sum(AM_svm_val))/12
LOGIT_val = (sum(O_log_val) + sum(cov_log_val) + sum(adult_log_val) +␣
 ↪sum(AM_log_val))/12
```

[4]:
```
KNN_val,SVM_val, LOGIT_val
```

[4]:
```
(0.8823500000000001, 0.8969333333333332, 0.8232833333333334)
```

[5]:
```
#Compute Stats for table1

adult_attr = len(adult_hot.columns) - 1
adult_tr = 5000
adult_tst = adult_hot.shape[0]-5000
adult_percentage = sum(adult_hot.income)/adult_hot.shape[0]

cov_attr = len(cov_bin.columns)-1
cov_tr = 5000
cov_tst = cov_bin.shape[0]-5000
cov_percentage = sum(cov_bin.class2)/cov_bin.shape[0]

lettersO_attr = len(letters_dat.columns)-3
lettersO_tr = 5000
lettersO_tst = letters_dat.shape[0]-5000
lettersO_percentage = sum(letters_dat.O_pos)/letters_dat.shape[0]

lettersAM_attr = len(letters_dat.columns)-3
lettersAM_tr = 5000
lettersAM_tst = letters_dat.shape[0]-5000
lettersAM_percentage = sum(letters_dat.AM_pos)/letters_dat.shape[0]
lettersAM_attr,lettersAM_tr,lettersAM_tst,lettersAM_percentage
```

```
      ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-5-204cdd2bb7b9> in <module>
```

```
      1 #Compute Stats for table1
      2
----> 3 adult_attr = len(adult_hot.columns) - 1
      4 adult_tr = 5000
      5 adult_tst = adult_hot.shape[0]-5000


NameError: name 'adult_hot' is not defined
```

[6]:
```
#Compute stats for table2

##knn
knn_acc = (sum(O_knn_acc) + sum(AM_knn_acc) + sum(cov_knn_acc) +␣
 ↪sum(adult_knn_acc))/12
knn_prec =␣
 ↪(sum(O_knn_prec)+sum(AM_knn_prec)+sum(cov_knn_prec)+sum(adult_knn_prec))/12
knn_recall = (sum(O_knn_recall)+sum(AM_knn_recall) +␣
 ↪sum(cov_knn_recall)+sum(adult_knn_recall))/12
knn_roc = (sum(O_knn_roc)+sum(AM_knn_roc)+sum(cov_knn_roc)+sum(adult_knn_roc))/
 ↪12
knn_f1 = (sum(O_knn_f1) + sum(AM_knn_f1)+ sum(cov_knn_f1)+sum(adult_knn_f1))/12
knn_mean = (knn_acc + knn_prec + knn_recall + knn_roc + knn_f1)/5

##svm

svm_acc = (sum(O_svm_acc)+sum(AM_svm_acc)+sum(cov_svm_acc)+sum(adult_svm_acc))/
 ↪12
svm_prec =␣
 ↪(sum(O_svm_prec)+sum(AM_svm_prec)+sum(cov_svm_prec)+sum(adult_svm_prec))/12
svm_recall = (sum(O_svm_recall)␣
 ↪+sum(AM_svm_recall)+sum(cov_svm_recall)+sum(adult_svm_recall))/12
svm_roc = (sum(O_svm_roc)+sum(AM_svm_roc)+sum(cov_svm_roc)+sum(adult_svm_roc))/
 ↪12
svm_f1 = (sum(O_svm_f1) + sum(AM_svm_f1)+ sum(cov_svm_f1)+sum(adult_svm_f1))/12
svm_mean = (svm_acc+svm_prec+svm_recall+svm_roc+svm_f1)/5

##logit

log_acc = (sum(O_log_acc)+sum(AM_log_acc)+sum(cov_log_acc)+sum(adult_log_acc))/
 ↪12
log_prec =␣
 ↪(sum(O_log_prec)+sum(AM_log_prec)+sum(cov_log_prec)+sum(adult_log_prec))/12
log_recall = (sum(O_log_recall)+sum(AM_log_recall) +␣
 ↪sum(cov_log_recall)+sum(adult_log_recall))/12
```

```
log_roc = (sum(O_log_roc)+sum(AM_log_roc)+sum(cov_log_roc)+sum(adult_log_roc))/
↪12
log_f1 = (sum(O_log_f1) + sum(AM_log_f1)+ sum(cov_log_f1)+sum(adult_log_f1))/12
log_mean = (log_acc + log_prec + log_recall + log_roc + knn_f1)/5
```

[7]:
```
##Compute stats for table3
```

[8]:
```
KNN_COV = sum((cov_knn_acc + cov_knn_prec + cov_knn_recall + cov_knn_roc +␣
↪cov_knn_f1)/5)/3
KNN_adult = sum((adult_knn_acc+ adult_knn_prec + adult_knn_recall +␣
↪adult_knn_roc + adult_knn_f1)/5)/3
KNN_letter1 = sum((O_knn_acc + O_knn_prec + O_knn_recall + O_knn_roc +␣
↪O_knn_f1)/5)/3
KNN_letter2 = sum((AM_knn_acc + AM_knn_prec + AM_knn_recall+O_knn_recall)/5)/3
KNN_mean = (KNN_COV + KNN_adult + KNN_letter1 + KNN_letter2)/4

SVM_COV = sum((cov_svm_acc + cov_svm_prec + cov_svm_recall + cov_svm_roc +␣
↪cov_svm_f1)/5)/3
SVM_adult = sum((adult_svm_acc+ adult_svm_prec + adult_svm_recall +␣
↪adult_svm_roc + adult_svm_f1)/5)/3
SVM_letter1 = sum((O_svm_acc + O_svm_prec + O_svm_recall + O_svm_roc +␣
↪O_svm_f1)/5)/3
SVM_letter2 = sum((AM_svm_acc + AM_svm_prec + AM_svm_recall+O_svm_recall)/5)/3
SVM_mean = (SVM_COV + SVM_adult + SVM_letter1 + SVM_letter2)/4

LOG_COV = sum((cov_log_acc + cov_log_prec + cov_log_recall + cov_log_roc +␣
↪cov_log_f1)/5)/3
LOG_adult = sum((adult_log_acc+ adult_log_prec + adult_log_recall +␣
↪adult_log_roc + adult_log_f1)/5)/3
LOG_letter1 = sum((O_log_acc + O_log_prec + O_log_recall + O_log_roc +␣
↪O_log_f1)/5)/3
LOG_letter2 = sum((AM_log_acc + AM_log_prec + AM_log_recall+O_log_recall)/5)/3
LOG_mean = (LOG_COV + LOG_adult + LOG_letter1 + LOG_letter2)/4
```

[26]:
```
#t test for table 3
```

[34]:
```
svm_cov_ba = (cov_svm_acc + cov_svm_prec + cov_svm_recall + cov_svm_roc +␣
↪cov_svm_f1)/5
svm_adult_ba = (adult_svm_acc+ adult_svm_prec + adult_svm_recall +␣
↪adult_svm_roc + adult_svm_f1)/5
svm_letter1 = (O_svm_acc + O_svm_prec + O_svm_recall + O_svm_roc + O_svm_f1)/5
svm_letter2 = (AM_svm_acc + AM_svm_prec + AM_svm_recall+O_svm_recall)/5

knn_cov_ba = (cov_knn_acc + cov_knn_prec + cov_knn_recall + cov_knn_roc +␣
↪cov_knn_f1)/5
```

```
knn_adult_ba = (adult_knn_acc+ adult_knn_prec + adult_knn_recall +␣
 ↪adult_knn_roc + adult_knn_f1)/5
knn_letter1_ba = (O_knn_acc + O_knn_prec + O_knn_recall + O_knn_roc + O_knn_f1)/
 ↪5
knn_letter2_ba = (AM_knn_acc + AM_knn_prec + AM_knn_recall+O_knn_recall)/5

log_cov_ba = (cov_log_acc + cov_log_prec + cov_log_recall + cov_log_roc +␣
 ↪cov_log_f1)/5
log_adult_ba = (adult_log_acc+ adult_log_prec + adult_log_recall +␣
 ↪adult_log_roc + adult_log_f1)/5
log_letter1_ba = (O_log_acc + O_log_prec + O_log_recall + O_log_roc + O_log_f1)/
 ↪5
log_letter2_ba = (AM_log_acc + AM_log_prec + AM_log_recall+O_log_recall)/5
```

[48]:
```
ttest_rel(knn_letter2_ba, svm_letter2)
ttest_rel(svm_letter2,knn_letter2_ba)
```

[48]:
```
Ttest_relResult(statistic=1.3481545484571051, pvalue=0.3100001731773808)
```

[40]:
```
svm_letter1
log_letter1_ba
```

[40]:
```
array([0.35748314, 0.36001735, 0.36128715])
```

[11]:
```
##t test for table 2

from scipy.stats import ttest_rel
```

[12]:
```
knn_acc_vec = np.concatenate((O_knn_acc, AM_knn_acc, cov_knn_acc,␣
 ↪adult_knn_acc))
svm_acc_vec = np.concatenate((O_svm_acc, AM_svm_acc, cov_svm_acc,␣
 ↪adult_svm_acc)) ##better than knn/logit
logit_acc_vec = np.concatenate((O_log_acc, AM_log_acc, cov_log_acc,␣
 ↪adult_log_acc))

knn_prec_vec = np.concatenate((O_knn_prec, AM_knn_prec, cov_knn_prec,␣
 ↪adult_knn_prec))
svm_prec_vec = np.concatenate((O_svm_prec, AM_svm_prec, cov_svm_prec,␣
 ↪adult_svm_prec)) ##better than knn/logit
logit_prec_vec = np.concatenate((O_log_prec, AM_log_prec, cov_log_prec,␣
 ↪adult_log_prec))

knn_recall_vec = np.concatenate((O_knn_recall, AM_knn_recall, cov_knn_recall,␣
 ↪adult_knn_recall))
svm_recall_vec = np.concatenate((O_svm_recall, AM_svm_recall, cov_svm_recall,␣
 ↪adult_svm_recall)) ##better than knn/logit
```

```
logit_recall_vec = np.concatenate((O_log_recall, AM_log_recall, cov_log_recall,␣
 ↪adult_log_recall))

knn_roc_vec = np.concatenate((O_knn_roc, AM_knn_roc, cov_knn_roc,␣
 ↪adult_knn_roc))
svm_roc_vec = np.concatenate((O_svm_roc, AM_svm_roc, cov_svm_roc,␣
 ↪adult_svm_roc)) #better than both
logit_roc_vec = np.concatenate((O_log_roc, AM_log_roc, cov_log_roc,␣
 ↪adult_log_roc))

knn_f1_vec = np.concatenate((O_knn_f1,AM_knn_f1,cov_knn_f1,adult_knn_f1))
svm_f1_vec = np.concatenate((O_svm_f1,AM_svm_f1,cov_svm_f1,adult_svm_f1))␣
 ↪#better than both
logit_f1_vec = np.concatenate((O_log_f1,AM_log_f1,cov_log_f1,adult_log_f1))
```

[25]:
```
## t_tests for table 2
```

[13]:
```
ttest_rel(svm_acc_vec, logit_acc_vec)
```

[13]:
```
Ttest_relResult(statistic=2.691412825053217, pvalue=0.020976129833102677)
```

[14]:
```
ttest_rel(svm_acc_vec, knn_acc_vec)
```

[14]:
```
Ttest_relResult(statistic=4.956872955702661, pvalue=0.00043091511348148975)
```

[15]:
```
ttest_rel(svm_prec_vec,knn_prec_vec)
```

[15]:
```
Ttest_relResult(statistic=3.899000623781768, pvalue=0.002480841886663993)
```

[16]:
```
ttest_rel(svm_prec_vec,logit_prec_vec)
```

[16]:
```
Ttest_relResult(statistic=2.724939353033039, pvalue=0.019756256264367063)
```

[17]:
```
ttest_rel(svm_recall_vec, knn_recall_vec)
```

[17]:
```
Ttest_relResult(statistic=2.4844287869016473, pvalue=0.03033415889370753)
```

[18]:
```
ttest_rel(svm_recall_vec,logit_recall_vec)
```

[18]:
```
Ttest_relResult(statistic=2.5921017215449846, pvalue=0.02504420630192683)
```

[19]:
```
ttest_rel(svm_roc_vec,knn_roc_vec)
```

[19]:
```
Ttest_relResult(statistic=5.407850247640286, pvalue=0.0002140905264235102)
```

[20]:
```
ttest_rel(svm_roc_vec,logit_roc_vec)
```

[20]:
```
Ttest_relResult(statistic=3.9692275385575546, pvalue=0.0021990063140027837)
```

[21]:
```
ttest_rel(svm_f1_vec,knn_f1_vec)
```

[21]:
```
Ttest_relResult(statistic=4.784996424186888, pvalue=0.0005668429198106821)
```

[22]:
```
ttest_rel(svm_f1_vec, logit_f1_vec)
```

[22]:
```
Ttest_relResult(statistic=2.645263577902502, pvalue=0.022778174295883264)
```

[23]:
```
knn_vec_tot = np.array([.885,.826,.777,.892,.795])
svm_vec_tot = np.array([.899,.850,.804,.936,.824])
log_vec_tot = np.array([.820,.546,.527,.842,.534])
```

[24]:
```
ttest_rel(svm_vec_tot,knn_vec_tot)
```

[24]:
```
Ttest_relResult(statistic=5.698296959205801, pvalue=0.004686824860794749)
```

[ ]:
```

```